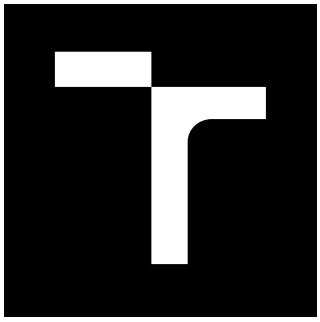
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

SEMESTRÁLNÍ PRÁCE

Brno, 2020 Lukáš Prusák



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

KNIHOVNA PRO PRÁCI SE SPECIFICKÝMI TDMS SOUBORY
LIBRARY FOR MANIPULATION WITH SPECIFIC TDMS FILES

SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE **Lukáš Prusák**
AUTHOR

VEDOUCÍ PRÁCE **Ing. Martin Čala**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Semestrální práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Lukáš Prusák ***ID:*** 203328
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Knihovna pro práci se specifickými TDMS soubory

POKYNY PRO VYPRACOVÁNÍ:

Tématem semestrální práce je vytvoření knihovny v LabVIEW, která bude sloužit k práci se specifickými TDMS soubory, zejména s nadměrně velkými (řádově 100 GB) nebo různě poškozenými, kde je cílem získat maximum korektních dat. Zadání shrnují následující body:

1. Popište formát souboru TDMS a obvyklé nástroje pro práci s ním.
2. S pomocí dodaných souborů navrhněte způsob, jak zkontrolovat správnost TDMS souboru.
3. Navrhněte a implementujte defragmentaci nadměrně velkých souborů.
4. Vytvořte nástroj pro generování poruch v TDMS souborech.

DOPORUČENÁ LITERATURA:

[1] TDMS File Format Internal Structure. National Instruments [online]. 2014-06-23 [cit. 2019-09-13]. Dostupné z: http://www.ni.com/product-documentation/5696/en/

Termín zadání: 23.9.2019 ***Termín odevzdání:*** 6.1.2020

Vedoucí práce: Ing. Martin Čala

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Cieľom tejto práce je vytvoriť knižnicu v prostredí LabVIEW, ktorá dokáže pracovať so špecifickými TDMS súbormi, s nadmernou veľkosťou alebo chybným formátom. V tejto práci sa knižnica zameriava hlavne na čítanie TDMS súboru, jeho defragmentáciu a spôsobmi, ktoré dokážu získať dáta z poškodených súborov tohoto typu. Hlavným prínosom tejto práce sú implementované funkcie v knižnici, ktoré môžu slúžiť na analýzu TDMS formátu, defragmentáciu nadmerne veľkých súborov a obnovu poškodených TDMS súborov.

KĽÚČOVÉ SLOVÁ

TDMS súbor, ukladanie dát, LabVIEW, čítanie súborov, defragmentácia, generovanie chýb, obnova súboru

ABSTRACT

The goal of this work is to create a library in LabVIEW enviroment that can work with specific TDMS files, with exessive size or malformed. In this work, the library focusees mainly on reading the TDMS file, its defragmentation and methods that can retrieve data from corrupted TDMS files. The main contribution of this work are implemented functions in the library, which can be used for analysis of TDMS format, defragmentation of oversized files and recovery of damaged TDMS files.

KEYWORDS

TDMS file, storing data, LabVIEW, reading files, defragmentation, generating of errors, file recovery

PRUSÁK, Lukáš. *Knihovna pro práci se specifickými TDMS soubory*. Brno, 2020, 66 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedúci práce: Ing. Martin Čala,

POĎAKOVANIE

Rád by som poďakoval vedúcemu bakalárskej práce Ing. Martinovi Čalovi, Ph.D. za odborné vedenie, konzultacie, trpezlivosť a podnetné návrhy k práci.

Brno 08.06.2020 .....

podpis autora

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Spôsob ukladania dát sa stáva čím ďalej tým dôležitejším prvkom pri vytváraní dizajnu hardvérových či softvérových systémov. Pri návrhu sa zameriavame na rôzne požiadavky, ktoré jednotlivé spôsoby ukladania dát dokážu splniť, ako napríklad priestor, aký dáta zaberajú, rýchlosť ich zápisu, či ich čitateľnosť. Niekedy je vhodné zvoliť spôsob, ktorý bude splňovať len niektoré požiadavky, aby sa zjednodušil návrh hardvéru či softvéru. No následkom dnešného pokroku vo výkone výpočtovej techniky a dopytu vedcov a inžinierov sa mnoho firiem snaží nájsť riešenie na univerzálnejší spôsob spracovania a formátovania dát. Samozrejme, väčšia univerzálnosť znamená aj väčšiu zložitosť týchto spôsobov, a preto je potrebné používať špeciálne softvérové aplikácie pre prácu s nimi.

Jednou z firiem, ktorá vytvorila spôsob ukladania dát, je National Instruments. Ich formát súborov, nazývaný TDMS, je hlavne zameraný na spracovanie nameraných dát meracích kariet, ktoré táto firma aj poskytuje, no je používaný taktiež pri rôznych iných spôsoboch merania dát. S TDMS formátom súborov sa dá pracovať v prostredí LabVIEW, a to pomocou predom vytvorených programoch nazývaných VI.

Táto práca sa zaoberá skúmaním TDMS súborov, objasnením spôsobu ich formátovania a vytváraním knižnice s rôznymi funkciami, ktoré umožnia prácu s týmto formátom. Knižnica by mala obsahovať v prvom rade nástroj, ktorý umožní grafickú analýzu TDMS formátu a pomocou neho získavať potrebné informácie pre návrh ostatných funkcii knižnice. Následne pomocou týchto informácií bude možné kontrolovať správnosť súborov tohto formátu a implementovať funkciu pre defragmentáciu nadmerne veľkých TDMS súborov, pri ktorých môže pôvodná funkcia v LabVIEW zlyhať.

Pri zapisovaní dát sa môže následkom poruchy súbor poškodiť a obvyklé funkcie v prostredí LabVIEW ich nedokážu otvoriť. Cieľom tejto práce je vytvoriť nástroj, ktorý by generoval takéto poruchy pre TDMS súbory. Následne pomocou reálnych a vygenerovaných súborov navrhnúť postup, ktorý by z nich získal maximum korektných dát.

1 Spôsoby ukladanie nameraných dát

Podľa typu aplikácie sa môžu uprednostňovať rôzne spôsoby. Najčastejšie sa vyskytujú formáty typu ASCII a binárne. Tieto spôsoby majú svoje výhody aj nevýhody v rôznych oblastiach ich využitia.

1.1 ASCII súbory

ASCII (American Standard Code for Information Interchange) je set 128 symbolov zakódovaných do 8 bitov, kde sa najvýznamnejší bit nepoužíva. Tento formát je dlho globálne zaužívaný a dokáže ho spracovávať akýkoľvek textový editor, pretože má veľmi malú zložitosť a jednoduchú čitateľnosť pre užívateľa. [1] ASCII formát obsahuje okrem klasických znakov aj kontrolné znaky, ako napríklad medzera, tabulátor či koniec textu, ktoré slúžia pre jeho formátovanie a následovne spracovanie zapísaných či nameraných údajov.

Jednoduchosť tohto zápisu dát má aj svoje nevýhody. Keďže ASCII ukladá všetko ako reťazec znakov, pričom dochádza ku konverzii dát, čo výsledne spolu s nevyužívaním najvyššieho bitu spôsobuje podstatne väčšiu veľkosť súborov na disku a taktiež dlhý zápis či čítanie dát, hlavne pri ukladaní nameraných dát, kde je mnoho čísel. Taktiež to znamená spomalenie následnej softvérovej analýzy dát a môže znižovať presnosť kvôli už spomínanej konverzii dát z desiatkovej na binárnu a naopak pri čítaní súborov.

Na preposielanie malého objemu dát je to vhodný formát, pretože osoba na druhej strane nebude pre čítanie potrebovať nijakú softvérovú nadstavbu, ale dokáže si zobraziť dáta v hocijakom vstavanom textovom editore. Naopak, tento formát je veľmi nevhodný pri meraní veľkého objemu dát, ktorý je potrebné ukladať, preposielať či opätovne softvérovo analyzovať. Pre takéto prípady je vhodnejšie zvoliť si binárny spôsob ukladania dát.

Znak	ASCII	Binárne
1	0011 0001	0000 0001
10	0011 0001 0110 0000	0000 1010
100	0011 0001 0011 0000 0011 0000	0110 0100
a	0110 0001	0110 0001
A	1000 0001	1000 0001

Obr. 1.1: Porovnanie ukladania dát v ASCII a v binárnych súboroch.

1.2 Binárne súbory

Využitie binárneho formátu je hlavne pri súboroch, ktoré nie sú zamerané iba na spracovanie textu. Porovnanie je možné vidieť na obrázku 1.1. pri kódovaní textu sa využíva rovnaký počet bajtov v binárnych aj v ASCII súboroch a zároveň sa stráca jeho čitateľnosť. Podstatný rozdiel je viditeľný pri kódovaní čísel, ktoré so zvyšujúcim sa počtom číslic šetria väčší počet bajtov. Výhody tohto formátu sa prejavujú pri veľkých objemoch dát kvôli jeho malému rozmeru na disku a vyšším rýchlostiam pri zápise a čítaní. Naopak, jeho nevýhodou je nečitateľnosť a problémovosť pri preposielaní. Pri práci je nutné poznať formát súboru a je potrebné používať softvérovo jednotne navrhnutý zápis a čítanie dát, keďže mnoho používateľov si vytvára vlastný zápis binárnych dát, ktorý nemusí fungovať správne pri rôznych verziách softvéru u rôznych používateľov. Preto sa používajú prípony pri súboroch (.jpg, .tex, .avi) pre ich zjednotenie a aby sa dokázal rozoznať ich formát a následne správne prečítať.

1.3 TDMS súbory

TDMS súbory patria medzi typ binárnych súborov špecializovaných na prácu s nameranými dátami. Tento formát súboru sa vyznačuje príponou **.tdms** a má vopred danú hierarchiu, ktorá je jednotná a umožňuje ľahkú strojovú čitateľnosť funkciami, ktoré sú vytvorené pre prácu s príponou .tdms. Taktiež sa v tejto hierarchii nachádza okrem dát aj hlavička, atribúty a iné popisné informácie, ktoré zvyšujú prehľadnosť a následnú rozšíriteľnosť súboru. Je možné vytvárať súbor v rovnakom formáte s príponou **.tdms_index**, kde sú uložené iba tzv. hlavičky súboru a to preto, aby sa dokázalo ľahšie a rýchlejšie pracovať s TDMS súbormi.

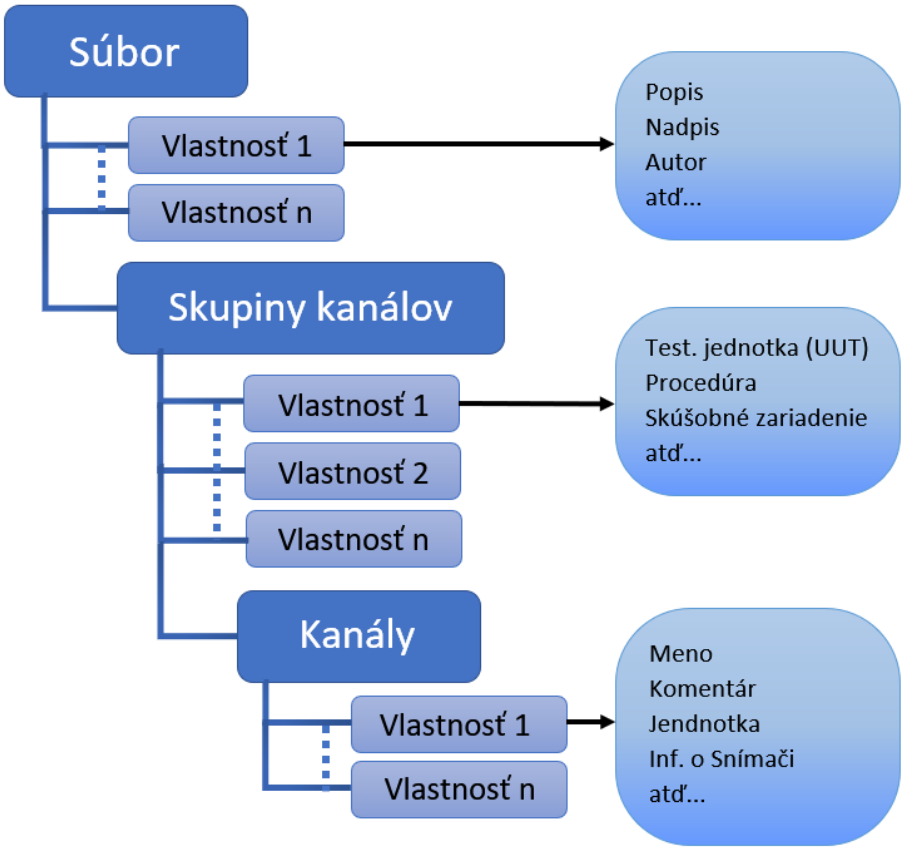
2 Hierarchia TDMS súborov

Keďže tento formát súboru je proprietárny, tak sa popisom jeho hierarchie venujú materiály od tvorcov - literatúra [2],[3].

TDMS súborový formát má tro-stupňovú hierarchiu, ako je možné vidieť na obrázku 2.1:

- súbor
- skupina (kanálov)
- kanál

Každý z týchto troch stupňov môže mať nekonečno vlastností, čo je vhodné na vytváranie hlavičiek. Taktiež každý súbor môže obsahovať nekonečno skupín a každá skupina môže obsahovať nekonečno kanálov. [3] To umožňuje prehľadné triedenie našich nameraných dát. S touto štruktúrou sa užívateľ stretne pri používaní štandardných nástrojov od National Instruments ako **LabVIEW** či **DIAdem**. Hierarchia súboru binárneho rozloženia sa líši a je popísaná v nasledujúcej podkapitole **2.1.**



Obr. 2.1: Hierarchia TDMS súboru (Inšpirované obrázkom z použitej literatúry [3]).

2.1 Binárne rozloženie

Binárne rozloženie TDMS sa trochu líši od jeho hierarchie. Pri každom zápise do súboru sa vytvorí nový **segment**, ktorý sa skladá z troch častí uložených za sebou v nasledujúcom poradí:

- **Lead In -** obsahuje informácie pre budúcu orientáciu sa v súbore a jeho spracovanie.
- **Meta data -** popisuje vyššie uvedenú hierarchiu súboru.
- **Raw data -** obsahuje vektor nameraných dát.

Tieto časti nie sú od seba oddelené alebo ich začiatok nie je vyznačený, preto sa na pohyb v súbore používajú ofsety, ktoré ukazujú na pozíciu jednotlivých častí.

tdsDataType	**Popis**
uint32 (4 B)	"TDSm"tag
uint32 (4 B)	ToC maska
uint32 (4 B)	Čislo verzia TDMS súboru (1.0, 2.0)
uint64 (8 B)	Ofset na ďalší segment
uint64 (8 B)	Ofset na Raw data

Tab. 2.1: Binárna štruktúra Lead In časti segmentu (Inšpirované tabuľkou z použitej literatúry [2]).

2.1.1 Lead In

Obsah Lead In časti je zobrazený v Tab.2.1, kde sú uvedené aj veľkosti jednotlivých informácií.

- Prvé 4 bajty každého segmentu sú používané pre **TDSm tag**, ktorý udáva jeho počiatok.
- Ďalšie 4 bajty sú používané pre **ToC masku**, ktorej bity v sebe uchovávajú informácie o tom, akého typu sú dáta alebo ako sú zapisované. Tabuľka pre ToC masku je zobrazená v tabuľke.2.2.

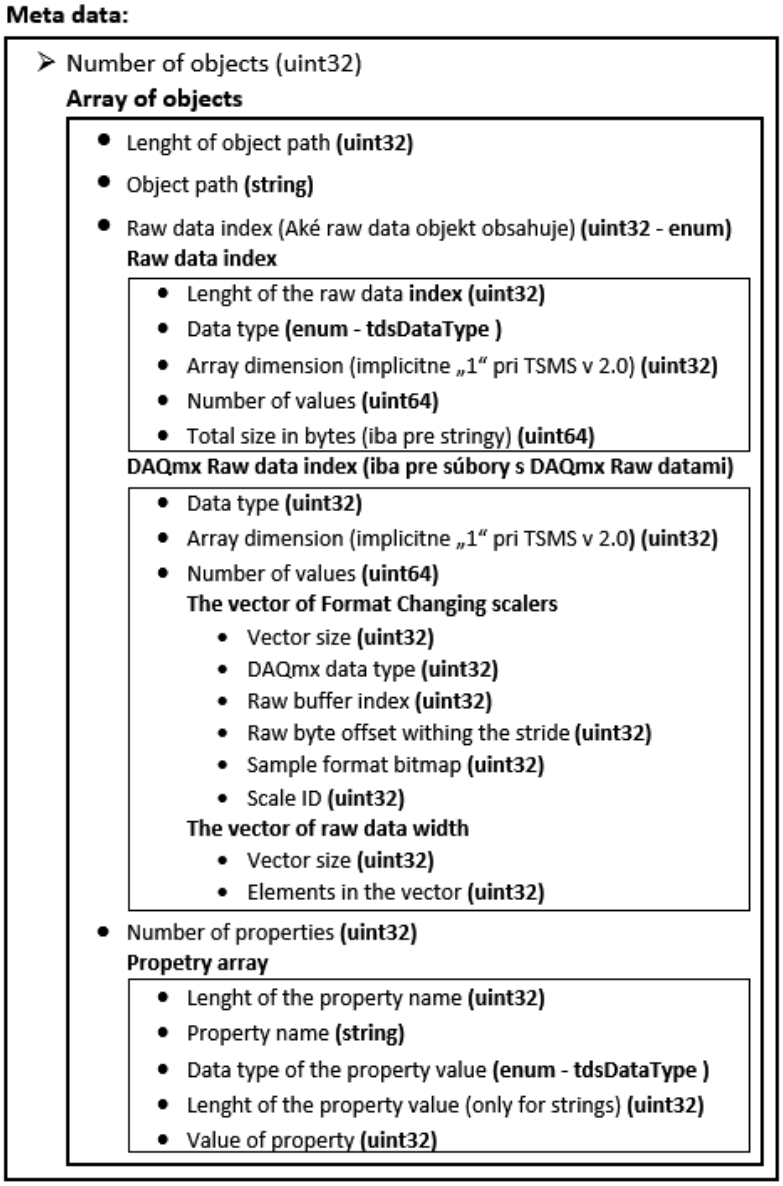
bit	Názov	Popis
1	kTocMetaData	Segment obsahuje Meta dáta
2	kTocNewObjList	Segment obsahuje nový objektový list (nový segment je odlišný od predošlého)
3	kTocRawData	Segment obsahuje Raw dáta
5	kTocInterleavedData	Raw dáta v segmente sú Interleaved
6	kTocBigEndian	Všetky číselné hodnoty v segmente (okrem "TDSm"tagu a ToC masky) sú formátované ako big endián
7	kTocDAQmxRawData	Segment obsahuje DAQmx Raw dáta

Tab. 2.2: ToC maska (Inšpirované tabuľkou z použitej literatúry [2]).

- Ďalšie 4 bajty popisujú verziu formátu TMDS súboru, ktorá môže nadobúdať 2 hodnoty `0x4712` pre formát TDMS súboru verzie 1.0 alebo `0x4713` formát TDMS súboru verzie 2.0, ktorá je teraz najaktuálnejšou verziou.
- Ďalších 8 bajtov obsahuje offset na nasledujúci segment alebo koniec súboru, ak sa v ňom už nenachádzajú ďalšie segmenty. Presnejšie popisuje dĺžku celého segmentu mínus dĺžka Lead In. Ak nastane pri zápise do TDMS súboru nejaký problém, ako napríklad výpadok napájania, tak všetky bajty tohto ofsetu sa nastavia na hodnotu `0xFF`, čo môže nastať iba v poslednom segmente TDMS súboru [2].
- Posledných 8 bajtov v časti Lead In obsahuje taktiež ofset, ktorý vyjadruje celkovú dĺžku Meta dát v segmente. Ak segment neobsahuje **Meta dáta**, táto hodnota bude rovná `0x00` [2]. Ak sa v segmente nenachádzajú **Raw dáta**, táto hodnota bude rovnaká ako **ofset na ďalší segment**.

2.1.2 Meta data

Meta data sa skladajú z objektov s hierarchiou, ktorá je zobrazená na obr.2.1. Podľa tejto hierarchie môže byť objektom **súbor**, **skupina** alebo **kanál**, čo je možné zistiť podľa jeho cesty. Iba kanály obsahujú namerané dáta, čiže **Raw dáta** 2.1.3. Každý objekt môže obsahovať určité množstvo vlastností. Binárna štruktúra dát je zobrazená obrázku 2.2. Jednotlivé názvy premenných v hierarchii sú v pôvodnom anglickom jazyku aby boli rovnaké ako v prostredí LabVIEW.



Obr. 2.2: Binárna hierarchia meta dát v súbore (Inšpirované tabuľkou z použitej literatúry [2]).

Meta dáta neobsahujú celú túto štruktúru, ale mení sa podľa toho, aké dáta sa do súboru zapíšu. Ak sa zapisujú dáta pomocou funkcie **TDMS Write** (bližšie opísané v podkapitole 3.1), Meta data nebudú obsahovať časť **DAQmx Raw data index parameters** obr. 2.2.

Ak sa budú dáta zapísovať pomocou nástrojov **DAQmx** (bližšie opísané v podkapitole 3.1), Meta data nebudú obsahovať časť **Raw data index parameters** (obr. 2.2).

To, aké parametre sa v Meta dátach nachádzajú, určuje **Raw data index**, ktorý je bližšie opísaný v podkapitole 4.2.1

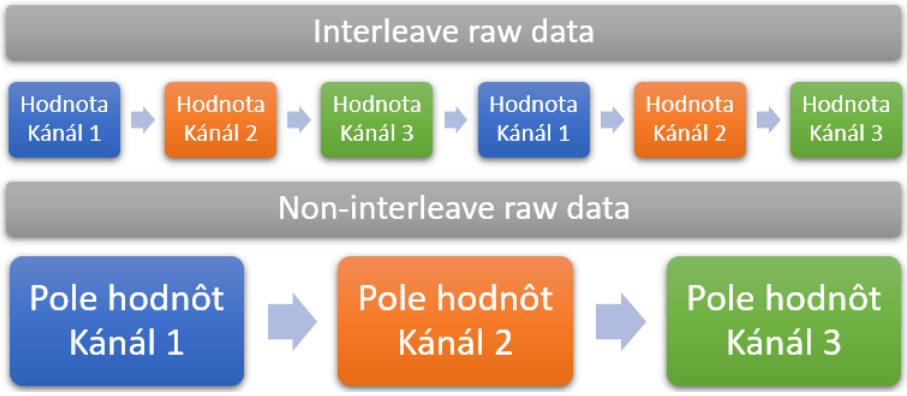
2.1.3 Raw data

Táto časť obsahuje Raw dáta pre každý kanál, ktoré sú zapísané ako dátové polia v rovnakom poradí, ako sú kanály uložené v časti segmentu Meta dáta [2].Ak sú raw dáta typu string, tak sa tiež zapisujú ako jeden spojitý celok, ale aby sme boli schopní oddeliť jednotlivé stringy, tak je potrebné vytvoriť pomocné pole ofsetov, ktoré sa bude nachádzať pred poľom stringov, podľa ktorého budeme vedieť presný začiatok a koniec jednotlivých stringov. Toto rozloženie umožňuje výber akéhokoľvek stringu zo súboru. Raw data majú dva typy rozloženia - „Interleaved“ alebo „Non-interleved“, ktoré je vopred určené v **ToC maske** v Lead In. Non-interleaved rozloženie Raw dát sa v LabVIEW nachádza pod názvom „decimed“.

Interleaved - ak máme viacej kanálov tak sa hodnoty v poli Raw dát zapisujú striedavo po jednej hodnote z každého kanálu.

Non-interleaved - všetky polia hodnôt, ktoré patria ku kanálom, sú zapísané ako spojitý celok a uložené v rovnakom poradí ako jednotlivé kanály.

Toto rozloženie je následne zobrazené na obr.2.3.



Obr. 2.3: Prekladanie raw dát, Interleave a Non-interleaved.

2.1.4 DAQmx Raw data

Je špeciálny typ dát, vytvorený pomocou DAQmx ovládača. Dáta sa ukladajú priamo z analógových alebo digitálnych výstupov na disk pomocou nástroja **DAQmx Configure Logging (TDMS)** v LabVIEW, ktorý je bližšie opísaný v podkapitole 3.1.

Tento typ Raw dát má samozrejme inú hierarchiu, preto sa v ToC maske nachádza bit, ktorý upozorňuje na výskyt DAQmx Raw dát v segmente.

TMDS súbor s DAQmx Raw dátami môžeme rozdeliť do troch častí:

1. segmenty obsahujúce informácie o meraných jednotkách a mierkach, pomocou ktorých sa prepočítavajú Raw dáta v súbore. Zvyčajne je to jeden segment,

ale ak je do súboru ukladaný aj údaj o čase, tvoria túto časť 2 segmenty. Prvý pre parametre času a druhý pre parametre kanálov.
2. segmenty obsahujúce Raw dáta.
3. segment obsahujúci informácie o čase a intervaloch, v ktorých boli hodnoty merané. Vlastnosti v tomto segmente slúžia na vytvorenie dátového typu "Waveform"v LabVIEW.

Pri skúmaní formátu vytvorených DAQmx súborov sa zistilo, že prvý segment tohto typu súboru, má veľkosť zaokrúhlenú na hodnotu 4 KB, pričom prebytočné bajty sú doplnená nulami. To môže spôsobiť problémy pri čítaní súboru, hlavne ak sa obsah súboru číta priamo za sebou.

Hlavný rozdiel oproti TDMS súboru s klasickými dátami je index Raw dát, kde je používaná iba hierarchia pre DAQmx Raw dáta, ktorá je zobrazená na obr. 2.2. Bližší popis čítania DAQmx Raw dát sa nachádza v podkapitole 4.2.1.

2.1.5 Dátové typy

Dátový typ raw dát je binárne uložený ako 4-bajtový unsigned. Aby sme zistili, o aký dátový typ, ide používame enum, v ktorom sa nachádzajú všetky možné dátové typy pre Raw dáta a Meta dáta TDMS súboru.

typedef enum {

tdsTypeVoid = 0,

tdsTypeI8,

tdsTypeI16,

tdsTypeI32,

tdsTypeI64,

tdsTypeU8,

tdsTypeU16,

tdsTypeU32,

tdsTypeU64,

tdsTypeSingleFloat,

tdsTypeDoubleFloat,

tdsTypeExtendedFloat,

tdsTypeSingleFloatWithUnit=0x19,

tdsTypeDoubleFloatWithUnit,

tdsTypeExtendedFloatWithUnit,

tdsTypeString=0x20,

tdsTypeBoolean=0x21,

tdsTypeTimeStamp=0x44,

tdsTypeFixedPoint=0x4F,

tdsTypeComplexSingleFloat=0x08000c,

tdsTypeComplexDoubleFloat=0x10000d,

tdsTypeDAQmxRawData=0xFFFFFFFF

} tdsDataType; [2]

3 Práca s TDMS súbormi

Pre prácu s TDMS súbormi existuje viacero možností, či už originálne nástroje vytvorené spoločnosťou National Instruments ako je **LabVIEW** a **DIAdem**, alebo nástroje vytvorené užívateľmi pomocou iných programov vďaka tomu, že formát TDMS súborov je verejne dostupný.

3.1 Nástroje v LabVIEW

V LabVIEW sú vytvorené funkcie, ktoré slúžia pre prácu s TDMS súbormi. Medzi základné patria TDMS Open, TMDS Write, TDMS Read atď. Tieto nástroje sú pri väčšine prípadov postačujúce. Ak je však potrebné pracovať s TDMS súbormi detailnejšie, v knižnici Advanced TDMS sa nachádzajú pokročilé nástroje, ktoré sa využívajú pre optimalizáciu rýchlosti zápisu alebo čítania TDMS súborov.

Všetky tieto nástroje pre prácu s TDMS súbormi v tejto kapitole fungujú správne len pod podmienkou, že TDMS súbory nie sú nijako poškodené a majú korektný formát.

TDMS Defragment V podkapitole 2.1 je ukázané, ako vyzerá binárne rozloženie TDMS súboru a akým spôsobom sa ukladajú do neho dáta. Keďže sa pri každom zápise vytvorí v súbore nový segment, ktorý obsahuje všetky 3 časti Lead In, Meta dáta a Raw dáta, často nastávajú situácie, keď do súboru zapisujeme mnohokrát bez toho, aby sme zmenili Meta dáta, čiže v podstate hlavičku súboru. Tieto Meta dáta, nám zbytočne zaberajú miesto bez toho, aby nám dodávali nejaké hodnotné informácie.

TDMS Defragment spracuje celý súbor tak, že pomocou .tdms_index súboru vyhľadá rovnaké Meta dáta a odstráni ich zo súboru, následne upraví ostatné časti, ako napríklad ofsety v časti Lead In, aby ukazovali na správne pozície a súbor bol znovu čitateľný. Taktiež všetky segmenty pretvorí tak aby, Raw dáta v nich boli Non-Interleaved.

Vďaka defragmentácii bude mať výsledný súbor iba jeden segment so všetkými potrebnými objektami s informačnou hodnotou rovnakou a zároveň bude zaberať menej miesta na disku.

Funkcia TDMS defrag zlyháva pri veľkých súboroch pravdepodobne kvôli vytváraniu záložnej kópie starého súboru, ktorá sa nemusí vojsť na systémový disk.

TDMS File Viewer Medzi pôvodné funkcie v prostredí LabVIEW patrí aj TDMS File Viewer, ktorý slúži na jednoduché prečítanie a zobrazenie TDMS súborov. Taktiež umožňuje prezeranie dát TDMS súboru graficky.

DAQmx Configure Logging (TDMS) Tento nástroj patrí do skupiny nástrojov DAQmx a slúži na zápis dát do TDMS súboru z **NI-DAQmx** ovládačov, či už hardvérových alebo simulovaných . Nástroj ukladá dáta a informácie o mierkach osobitne, čo minimalizuje veľkosť súboru. Taktiež nepotrebuje byť defragmentovaný vďaka spôsobu ukladania dát, preto pokus o defragmentáciu pomocou nástroja **TDMS Defragment** vedie k chybe [7].

3.2 Program DIAdem

DIAdem je softvér pre spracovávanie a analýzu dát nazbieraných meraním alebo vygenerovaných simuláciou. [4] DIAdem je špeciálne navrhnutý pre prácu s TDMS súbormi, ale taktiež dokáže pracovať aj s vlastnými formátmi súborov, ak sa využije DataPlugin.

V programe je možné jednoducho TDMS súbor otvoriť, kde je následne poskytnutá jeho hierarchia, ako je vysvetlená v kapitole 2 na obrázku. 2.1.

V záložke **VIEW** sú na výber kanály s Raw dátami, ktoré je možné prezerať pomocou tabuliek alebo aj graficky.

Ak je na spracovanie dát matematická analýza, v záložke **ANALYSIS** sú dostupné rôzne funkcie pre spracovanie, napríklad sumácia, integrácia, štatistiky, ale aj zložitejšie ako analýza nárazov.

Po úspešnom spracovaní dát je možné tieto výsledky pomocou záložky **REPORT** graficky vyobraziť a predať zvyšku personálu v inštitúcii pre jednoduchšie prezentovanie dosiahnutých výsledkov.

Ak je pre spracovanie dát potrebné vykonávať rovnaké úkony, v nasledujúcej záložke **SCRIPT** je možné napísať skript pre zautomatizovanie týchto úkonov. Skript sa nemusí písať len ručne, ale môže sa využiť funkcia pre nahrávanie úkonov, ktoré je potrebné vykonať iba prvýkrát, funkcia si ich zapamätá a bude ich vykonávať za vás.

DIAdem je vhodný nástroj pre rýchlu prácu alebo len náhľad TDMS súborov vďaka tomu, že nie je nič nutné programovať.

3.3 Užívateľské nástroje

Keďže štruktúra TDMS súborov je zverejnená zakladateľom National Intruments a nie každý užívateľ, ktorý chce pracovať s týmto formátom je zároveň aj používateľom LabVIEW, tak vznikli viaceré nástroje pre čítanie tohto formátu v iných programovacích jazykoch.

TDMS Read v Matlabe

Aj pre Matlab bola vytvorená knižnica, ktorá dokáže čítať obe verzie tohto formátu bez potreby nainštalovať si DLL súbory od National Instruments. Taktiež podporuje čítanie Interleaved Raw dát a funguje vo všetkých verziách Matlabu. [5]

TDMS Read v pythone

Je to balíček pre jazyk python, ktorý dokáže čítať a zapisovať TDMS súbori a je postavený na tzv. numpy balíčku. To znamená, že prečítané dáta a dáta pri zapisovaní sú uložené v numpy poliach, čo sú polia, ktoré sú indexované v každej dimenzii. [6]

4 Čítanie TDMS súborov

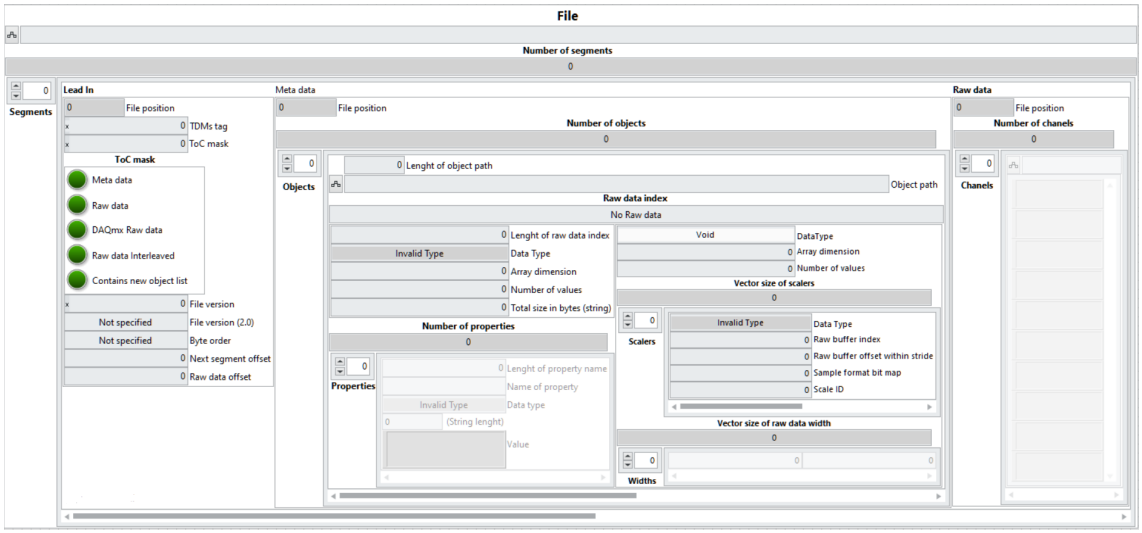
Pôvodné nástroje v LabVIEW v podkapitole 3.1 umožňujú prečítať iba dáta, ktoré sa nachádzajú v súbore, no nezobrazujú spôsob akým boli dáta uložené. Preto na obyčajné čítanie dát sú tieto nástroje postačujúce. V tejto práci je prednejšia analýza štruktúry súboru, preto je potrebné vytvorenie funkcie, ktorá zobrazí akým spôsobom sa vytvárajú rôzne typy TDMS súborov, čo bude veľmi nápomocné pri tvorení ďalších nástrojov.

V prostredí LabVIEW sa ako pri každom čítaní alebo zapisovaní musí najprv súbor otvoriť a následne je s ním možné pracovať pomocou jeho referencie. To sa vykonáva samostatne pomocou funkcie `Open/Create/Replace File` už implementovanej v LabVIEW, ktorá umožní binárne čítanie súboru.

Pred vytvorením funkcie na čítanie súboru je najprv potrebné vytvoriť štruktúru, do ktorej sa budú ukladať dáta prečítané zo súboru. Táto štruktúra je zobrazená na obr. 4.1 a je zhotovená v prostredí LabVIEW. Štruktúra sa skladá z častí:

- **cesta/referencia súboru**
- **počet segmentov**
- **štruktúra TDMS súboru** (kap. 2.1)

Táto štruktúra tak obsahuje všetky potrebné informácie o súbore a je možné s ňou pracovať aj v iných funkciách.

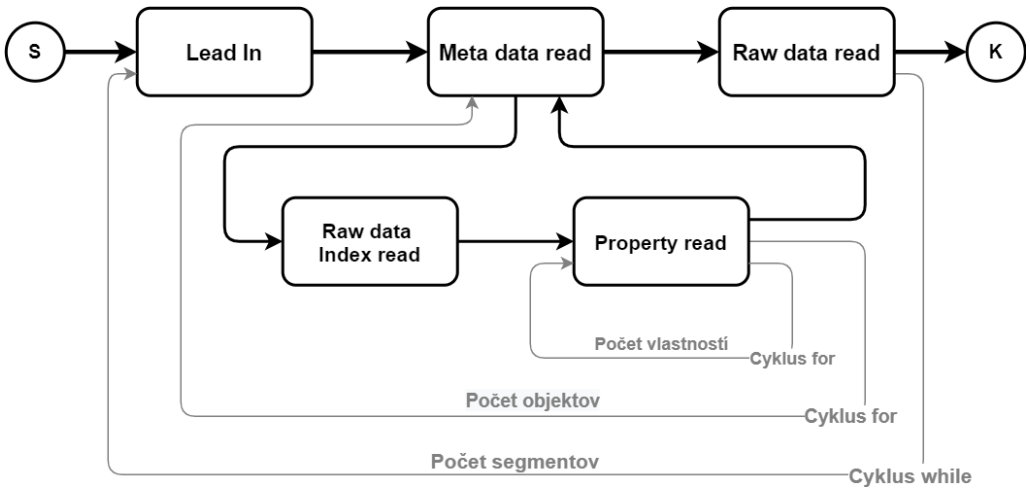


Obr. 4.1: Celá štruktúra TDMS súboru v prostredí LabVIEW.

Pred začatím procesu čítania súboru je vhodné skontrolovať, či je súbor bezchybný a najjednoduchší spôsob, ako to urobiť, je pokúsiť sa ho otvoriť pomocou pôvodných funkcií v LabVIEW(**TDMS Open, TDMS Close**) 3.1. Ak nastane pri otváraní súboru chyba, program môže upozorniť užívateľa. To je však možné

iba vtedy, ak je funkcia vytváraná v prostredí LabVIEW. O ďalšej kontrole TDMS súboru je napísané viac v podkapitole 7.2.

Po kontrole súboru je možné začať s jeho čítaním. Podľa vyššie uvedenej hierarchie 4.1 sa dá vytvoriť vývojový diagram algoritmu, ktorý je zobrazený na obr. 4.2



Obr. 4.2: Vývojový diagram algoritmu pre čítanie TDMS súborov.

Ako štruktúra súboru, tak aj čítanie je rozdelené do troch hlavných procesov, ktoré môžu byť pre optimalizáciu rýchlosti čítania paralelizované (napr. pipelining):

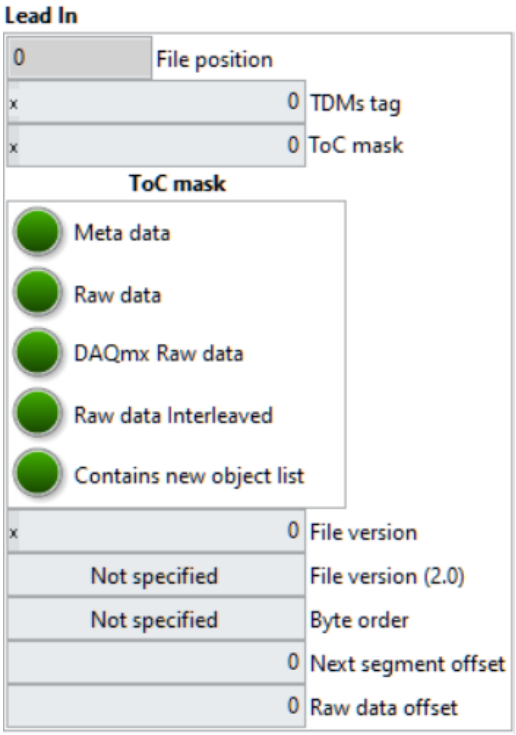
- **Lead In read** kap. 4.1
- **Meta data read** kap. 4.2
- **Raw data read** kap 4.3

Z týchto častí sa skladá jeden segment súboru, preto tieto procesy je pre prečítanie celého súboru nutné opakovať pre všetky segmenty, ktoré sa v súbore nachádzajú. Po prečítaní celého súboru sa počet segmentov uloží, ako je uvedené na obr. 4.1.

Pri každej z týchto častí sa nachádza aj pozícia ich počiatku v súbore pre lepšiu orientáciu či možnú kontrolu správnosti súboru, ale nie sú potrebné pre funkčnosť čítania. Zisťovanie týchto pozícií sa vykonáva vo funkcií **Lead In read** 4.1.

4.1 Lead In read

Táto funkcia má za úlohu pre čítať Lead In časť segmentu, ktorá je najjednoduchšia z týchto troch častí, keďže veľkosť Lead In časti je stále 28 bajtov, ktoré je vhodné prečítať naraz, a potom s ním pracovať a ukladať jednotlivé hodnoty do štruktúry uvedenej na obr. 4.3 pre optimalizovanie rýchlosti čítania súboru.



Obr. 4.3: Štruktúra Lead In v prostredí LabVIEW - výrez z obr. 4.1 (V tejto štruktúre sú niektoré hodnoty zobrazené 2-krát vo formáte, v akom sa nachádzajú v súbore a následne tak, aby bolo jasné, čo hodnoty znamenajú. V súbore sú zapísané iba raz).

TDSm tag a ToC maska majú stále bajtové poradie **little endian**, preto je možné Lead In čítať s týmto bajtovým poradím až po koniec ToC masky, čo znamená prvé dve 4 bajtové hodnoty (uint32). Ďalej v súbore sa už používa bajtové poradie, ktoré sa nachádza v ToC maske, takže zvyšok Lead In sa bude čítať už s týmto bajtovým poradím. Toto bajtové poradie sa už v súbore nemení, takže ho môžeme používať pre celý súbor.

Z pozície v súbore a z prečítaných ofsetov sa môžu uložiť pozície jednotlivých častí, a taktiež načítať pole bajtov pre ďalšie časti. Veľkosť poľa sa rovná hodnote ofsetu na ďalší segment. Toto pole sa pošle do funkcie **Meta data read**, ktorá z neho vyčíta potrebné údaje, a zostatok poľa pošle do funkcie **Raw data read**.

Po prečítaní tohto poľa je nutné skontrolovať, či sa aktuálna pozícia v súbore nenachádza na jeho konci a prípadne ukončiť ďalšie opakovanie cyklu.

Taktiež je vhodné obmedziť veľkosť súboru, ktorý sa dokáže prečítať, aby doba načítania nebola vysoká a program sa neukončil s chybou nedostatku pamäte. To je možné dosiahnuť napríklad obmedzením počtu segmentov alebo ukončením po presiahnutí určitej veľkosti súboru, alebo kombináciou oboch, ktorý je použitý v tejto práci.

Z pozície v súbore a indexu cyklu pre počet segmentov je možné vypočítať priemernú veľkosť segmentu a následne ju pripočítať k aktuálnej pozícii, čo umožní čiastočné predvídanie veľkosti po načítaní ďalšieho segmentu. Ak táto hodnota prekročí vopred zadanú hodnotu, proces čítania sa ukončí. Taktiež je vhodné, aby sa dalo zistiť ,akým spôsobom sa čítanie ukončilo.

4.2 Meta data read

Ich čítanie sa vykonáva podľa štruktúry, ktorá je uvedená na obr. 2.2 a obr. 4.1. Každé Meta dáta v sebe obsahujú pole objektov. Prvý údaj je stále ich počet, ktorý môže znamenať počet nových alebo zmenených objektov.

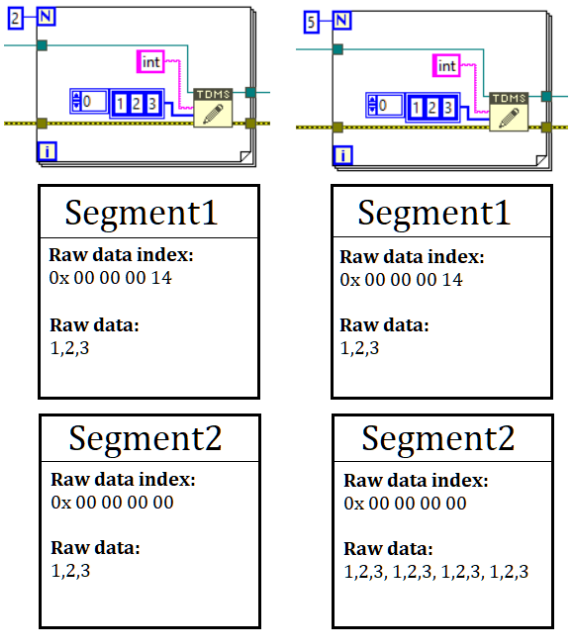
Každý objekt má svoju **cestu**, ktorá sa číta tak, že pred ňou v súbore sa nachádza jej dĺžka a podľa toho vieme prečítať celú cestu, ktorá je uložená v ASCII znakoch. Takýmto spôsobom sa v súbore číta každá cesta alebo string.

Ďalej sa v súbore nachádza **index Raw dát** a **vlastnosti objektu**.

4.2.1 Raw data index read

Raw data index je odlišný pre DAQmx dáta, preto prvé 4 bajty nám určia, čo budeme čítať. Celkovo môže nastať 5 rôznych možností, akými sa bude začínať Raw data index. (Tieto hodnoty sa taktiež menia s bajtovým poradím)

- `0x FF FF FF FF` - objekt neobsahuje Raw dáta. To znamená, že už ďalej nie je potrebné čítať Raw dáta index a je možné pokračovať v čítaní súboru ďalej.
- `0x 69 12 00 00` - Raw data typu DAQmx Format Changing scaler. Dáta, ktoré sú čítané z analógových vstupov DAQmx kariet.
- `0x 69 13 00 00 / 0x 6A 12 00 00` - Raw data typu DAQmx Digital Line scaler. Dáta, ktoré sú čítané z digitálnych vstupov DAQmx kariet. (V literatúre [3] sa uvádza hodnota bajtov: `0x 69 13 00 00`, ale vo vytvorených súboroch je táto hodnota: `0x 6A 12 00 00`)
- `0x 00 00 00 00` - index Raw dát sa rovná indexu z rovnakého kanálu v niektorom z predošlých segmentov. V TDMS verzií 2.0 sa zapisovanie do súboru pomocou cyklu **for** správa tak, ako je zobrazené na obr.4.4. To môže nastať iba vtedy, ak všetky kanály v aktuálnom segmente majú rovnakú hodnotu indexu Raw dát (`0x 00 00 00 00`). Vo verzii 1.0 sa pri každom zápise vytvorí nový segment. Podrobnejšie vysvetlené v podkapitole 5.1.3.



Obr. 4.4: Spôsob uloženia TDMS súboru, pokiaľ sa dáta opakovanie zapisujú v cykle. Segment 1 obsahuje vždy hodnoty iba z prvého zápisu, zbytok dát sa nachádza v Segmente 2 s indexom Raw dát `0x 00 00 00 00` pre verziu 2.0) .

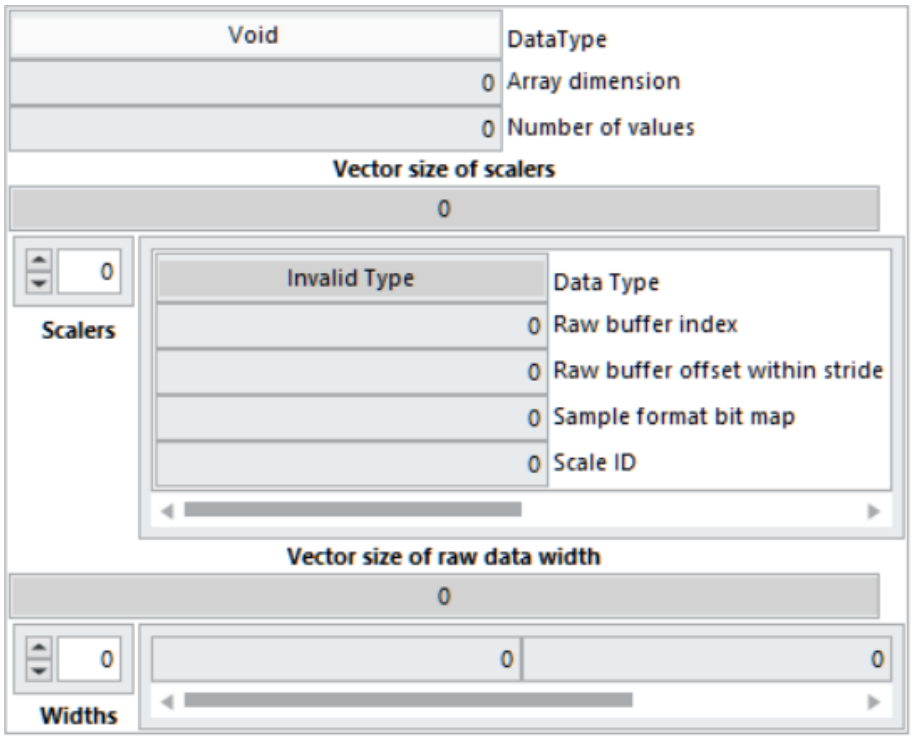
Vo verzii 1.0 sa vytvorí počet segmentov ekvivalentný počtu opakovaní cyklu.

- Ak žiadna z predošlých možností nenastane, znamená to,že sa prečítala dĺžka nového indexu Raw dát, čo je počiatočná hodnota parametrov indexu Raw dát. Ďalšie parametre, ktoré nasledujú v indexe Raw dát, sú uvedené na obr. 2.2.

Počas tohto procesu čítania Raw data indexu je možné inicializovať pole kanálov pre Raw dáta. Ak objekt obsahuje akýkoľvek index Raw dat okrem hodnoty `0x FF FF FF FF`, znamená to, že tento objekt je kanál. Preto je vhodné jeho cestu alebo iba koniec jeho cesty vložiť do poľa kanálov, a tak ho inicializovať, aby pri zobrazovaní kanálov mohol užívateľ vidieť, o aký kanál ide. Taktiež je vhodné si výsledný počet kanálov uložiť, tento údaj sa ďalej zíde pri čítaní Raw dát v ďalšej podkapitole 4.3.

DAQmx Raw data index

Index DAQmx Raw dát má svoju vlastnú štruktúru ,ako už bolo naznačené v kapitole 2. Pre lepšiu prehľadnosť je štruktúra pre index **analógových DAQmx Raw dát** (DAQmx Format Changing scaler) zobrazená aj na obr. 4.5.



Obr. 4.5: Štruktúra indexu pre analógové DAQmx Raw dáta v prostredí LabVIEW - výrez z obr. 4.1.

Štruktúra pre index **digitálnych DAQmx Raw dát** (DAQmx Digital Line scaler) je rovnaká až na parametre **Vektora veľkostí mierok** (Vector size of scalers), obr. 4.5. Podľa binárnych dát sa zistilo, že sa namiesto týchto parametrov sa stále nachádza 17 bajtov, ktoré majú takmer stále nulové hodnoty. Keďže formát indexu digitálnych DAQmx Raw dát je v literatúre [3] iba povrchne a o týchto hodnotách nie je žiadna zmienka, nepodarilo sa zistiť ich presnejší význam. Tieto parametre ale nie sú potrebné pre čítanie Raw dát, ako je tomu pri čítaní klasických dát.

Keďže zostatok štruktúry je rovnaký pre obe type DAQmx Raw dát, plnia rovnaký účel:

- **dátový typ** (DataType) - pri DAQmx Raw dátach musí nadobúdať hodnotu `0x FF FF FF FF`, čo je aj hodnota pre DAQmx Raw dáta uvedená v podkapitole pre dátové typy 2.1.5.
- **dimenzia poľa** (Array Dimension) - vo verzii 2.0 má stále hodnotu 1.
- **počet hodnôt** (Number of Values) - aj keď podľa názvu premennej sa zdá, že hodnota udáva počet hodnôt uložených v Raw dátach, nie je tomu tak. Reálny počet hodnôt je možné vypočítať vydelením celkovej veľkosti poľa s Raw dátami s veľkosťou jedného prvku.
- **vektor mierok** (The vector of scalers)- ani pri jednom type sa nepodarilo zistiť význam týchto parametrov, keďže nie sú nijako opísané v literatúre a vo

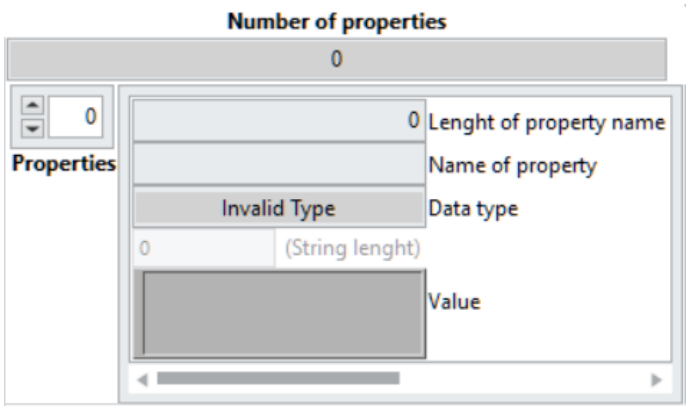
väčšine prípadov nadobúdajú nulové hodnoty. Taktiež dátový typ podľa enumu uvedeného v podkapitole 2.1.5 nesedí s reálnym dátovým typom použitým v TDMS súbore.

- **vektor šírky Raw dát** (The vector of Raw data width)- je jediný parameter, ktorý je potrebný pre ďalšie čítanie Raw dát. Prvý prvok tohto vektora môže určovať buď veľkosť jedného prvku v poli Raw dát, alebo veľkosť prvých prvkov všetkých kanálov (**veľkosť prvku x počet kanálov**). Vyjadrenie tohoto parametra závisí na DAQmx zariadení. Typ súboru, ktorý by mal viac ako len jeden prvok, sa nepodarilo vytvoriť, čiže sa predpokladá, že takýto prípad nenastane.

Všetky parametre, ktoré bližšie popisujú spracovávanie DAQmx Raw dát, sú uložené v predošlom segmente v príslušných objektoch ako **vlastnosti**, ktorých čítaním sa zaoberá ďalšia časť kapitoly 4.2.2.

4.2.2 Property read

Táto funkcia slúži na čítanie vlastností jednotlivých objektov. Tieto vlastnosti sú uložené v štruktúre zobrazenej na obr. 4.6.



Obr. 4.6: Štruktúra vlastností jednotlivých objektov v prostredí LabVIEW - výrez z obr. 4.1.

Každý objekt môže mať určitý počet vlastností. Hodnota s ich počtom sa nachádza hneď za Raw dáta indexom. Následne je tieto vlastnosti možné čítať a ukladať do poľa vlastností. Ďalší v poradí je dátový typ, ktorý určí následné čítanie hodnoty pre aktuálnu vlastnosť. Numerické dátové typy majú svoju danú dĺžku v bajtoch, no ako už bolo spomenuté, dátový typ string má pred sebou 4 bajtové číslo určujúce jeho dĺžku.

4.2.3 Kontrola zoznamu objektov

Prípad opisovaný v podkapitole 4.2.1, kde Raw dáta sú zapísané v segmente viacnásobne, môže nastať aj keď segment neobsahuje nový zoznam objektov, čo nám určuje hodnota z ToC masky **„kTocNewObjList“** zobrazená v tabuľke 2.2.

Táto hodnota hovorí o tom, že v segmente sa nachádzajú všetky objekty z predošlého segmentu, ale v Meta dátach sa nachádzajú iba objekty, ktorých hodnoty boli pozmenené.

Pri skúmaní vytvorených súborov sa zistilo, že tento prípad nastáva, ak sa pri zápise do súboru v cykle mení parameter jedného z kanálov (napr. celková veľkosť kanálu pre dátový typ string), a preto aj v tomto prípade môže nastať viacnásobné zapisovanie Raw dát do súboru podobne ako na obrázku 4.4

Z týchto dôvodov a z dôvodov uvedených v predošlej podkapitole 4.2.1, keď index Raw dát všetkých kanálov nadobúda hodnotu `0x 00 00 00 00`, je nutné pri čítaní Meta dát každého segmentu ukladať zoznam objektov a prípadne pri ich opakovanom výskyte aktualizovať ich hodnoty . To umožní správne prečítanie Raw dát v segmente, čím sa zaoberá ďalšia podkapitola 4.3.

4.3 Raw data read

Pri čítaním Raw dát je nutné používať predošle uložené informácie zo segmentu. Z ToC masky je možné zistiť, či sa v segmente Raw dáta nachádzajú, ale pri čítaní DAQmx dát to nemusí fungovať. Pri DAQmx Raw dátach ToC maska stále vyjadruje, že sa v segmente Raw dáta nachádzajú, ale neplatí to v prvom segmente. Kvôli dôvodom uvedením v podkapitole 2.1.4 je vhodnejšie porovnávať hodnoty ofsetov z Lead In. Ak sa ofset ďalšieho segmentu rovná ofsetu na Raw dáta, znamená to, že sa v segmente Raw dáta nenachádzajú. Ak sa táto podmienka nesplní, v segmente sú prítomné Raw dáta.

Raw dáta je možné čítatať postupným prechádzaním objektov a čítaním ich indexu Raw dát sa zisťuje, či daný objekt má Raw data a akého sú typu. Ak súbor obsahuje Raw dáta nie typu DAQmx , ktorých postup čítania si vysvetlíme v ďalšej časti kapitoly 4.3.1, môže ich program z načítaného poľa bajtov uložiť. Pomocou parametrov indexu Raw dát **dátového typu** a **počtu hodnôt** prečíta dáta. Rozdiel medzi **interleaved** a **non-interleaved** nastáva až pri ukladaní. Údaj o tom, v akom rozložení sú Raw dáta, získame z ToC masky:

- **non-interleaved** dáta je možné hneď ukladať do jednotlivých kanálov.
- **interleaved** Raw dáta sa zapisujú do TDMS súborov za pomoci 2D polí. To znamená, že všetky hodnoty budú rovnakého dátového typu a budú mať rovnaký počet hodnôt. Taktiež v prípade, ak je súbor defragmentovaný funkciou

TDMS Defragment, nemôže nastať, že kanály budú obsahovať rôzny počet hodnôt a zároveň budú mať Raw dáta s rozložením **interleaved** vďaka tomu, že táto funkcia ukladá Raw dáta automaticky v non-interleaved rozložení, čo je presnejšie vysvetlené v podkapitole 3.1. To nám umožní čítať interleaved dáta omnoho jednoduchšie. Dáta sa najprv uložia do 2D poľa, ktoré je potrebné pred uložením do kanálov transponovať.

Taktiež sa nesmie zabudnúť prípady keď sú Raw dáta v súbore zapísané viac násobne. Najjednoduchší spôsob ako prečítať všetky Raw data, je kontrolovať veľkosť načítaného poľa bajtov. Ak sa veľkosť poľa po skončení čítania nerovná 0 nemusí to znamenať chybu ale to, že neboli prečítané všetky Raw data, preto je potrebné zopakovať celý proces čítania Raw dát pomocou cyklu while. Až potom je možné Interleaved dáta transponovať a uložiť do príslušných kanálov.

4.3.1 DAQmx Raw data read

Čítanie DAQmx Raw dát sa veľmi nelíši od čítania klasických dátových typov. Počet hodnôt a ich veľkosť je možné zistiť z hodnôt ofsetov nachádzajúcich sa v Lead In a z prvej hodnoty vektora šírky Raw dát ako ,je popísané následovne:

- ***veľkosť segmentu*** = *ofset na ďalší segment - ofset na Raw dáta*
- ***veľkosť prvku*** = *vektor šírky Raw dát(0) / počet kanálov*

  (iba v prípade, ak *vektor šírky Raw dát(0) > počet kanálov*, výsledok musí vyjsť bez zvyšku, inak niekde nastala chyba)
  (počet kanálov, ktorý bol uložený pri čítaní Meta dát v podkapitole 4.2)

Pomocou týchto údajov je možné prečítať Raw dáta, no tieto hodnoty však nemusia odpovedať reálne nameraným hodnotám. Rozdielne je nutné riešiť, ak ide o digitálne alebo analógové DAQmx Raw dáta, čo nám určuje index Raw dát: (Pri segmentoch, ktoré obsahujú DAQmx Raw dáta, nemôže nastať prípad, kde by index Raw dát obsahoval hodnotu `0x 00 00 00 00`).

Digitálne DAQmx Raw dáta (DAQmx Digital Line scaler)

Výsledné hodnoty po prečítaní by mali dosahovať iba hodnoty „0“ a „1“. V niektorých prípadoch to hneď po prečítaní Raw dát tak nie je (napr. pri simulovaných DAQmx zariadeniach). Skutočne nameranej hodnote zodpovedá posledný bit hodnoty v Raw dátach.

Analógové DAQmx Raw dáta (DAQmx Format Changing scaler)

V tomto prípade je Raw dáta potrebné prepočítať pomocou mierky. Tieto mierky pre jednotlivé kanály sa nachádzajú v predošlom segmente, ako bolo popísané v

podkapitole 2.1.4. V predošlom segmente sú uložené všetky objekty, nielen kanály, ktoré sú ukladané stále ako posledné v poli. Na získanie mierok sú potrebné iba **vlastnosti** kanálov, preto pri čítaní predošlého segmentu je vhodné odstrániť všetky objekty, ktoré nie sú kanálmi. To je jednoduché, ak je počet kanálov zistený počas čítania Meta dát. Pole kanálov je potrebné až pri čítaní Raw dát v ďalšom segmente.

Na prepočet Raw dát pomocou mierky budú potrebné určité vlastnosti priradené ku kanálom:

[1]

1. „`NI_Scale[1]_Scale_Type"`[2] - vlastnosť dátového typu **string**, ktorá vyjadruje typ úpravy mierky. Vo všetkých vytvorených súboroch sa objavil iba typ "Linear", čo znamená lineárne prepočítavanie hodnôt pomocou známej rovnice 4.1

$$y = kx + q \tag{4.1}$$

2. „`NI_Scale[1]_Linear_Slope"`[3] - smernica k pre rovnicu 4.1, dátového typu **double**.
3. „`NI_Scale[1]_Linear_Y_Intercept"`[4] - posunutie q pre rovnicu 4.1, dátového typu **double**.

Je teda zrejmé, že výsledné hodnoty budú taktiež dátového typu double.

DAQmx Raw dáta sú takmer výhradné v rozložení **Interleaved**, preto je nutné ich pred prepočtom mierok uložiť do 2D poľa a transponovať, pretože každý kanál môže mať iné hodnoty vlastností, čiže aj iné parametre rovnice 4.1.

[1] hodnoty v hranatých zátvorkách [i], za názvom hovoria o pozícií v poli vlastností

5 Návrh a implementácia defragmentácie TDMS súborov

Pôvodná funkcia TDMS Defragment, ktorá bola opísaná v podkapitole 3.1 funguje tak, že pôvodný súbor sa prerobí na defragmentovaný, a pritom si uchová kópia pôvodného súboru počas defragmentácie, aby v prípade chyby pri defragmentácii dokázala súbor obnoviť a nestratiť žiadne dáta. To môže počas doby defragmentácie zabrať na disku veľa priestoru a pri veľkých súboroch dokonca ukončenie defragmentácie z dôvodu nedostatočného priestoru na disku. Táto chyba je dôvodom, prečo sa táto práca bude defragmentáciou TDMS súboru zaoberať.

Tomuto problému je možné sa vyhnúť len s úvahou, že defragmentovaný súbor je podstatne menší ako pôvodný. Potom je možné vytvoriť samostatný defragmentovaný súbor a prípadne vymazať pôvodný súbor až po dokončení defragmentácie. Zároveň pôvodný súbor môže slúžiť aj ako kontrola správneho procesu defragmentácie počas vývoja funkcie.

Proces sa vo svojej podstate veľmi nelíši od procesu čítania popísanému v kapitole 4, rozdiel je v účeloch týchto procesov. Táto kapitola sa bude venovať hlavne optimalizácii procesu s ohľadom na pamäť a rýchlosť, a to kvôli zameraniu knižnice na veľké TDMS súbory.

Ako už bolo spomenuté je potrebné najprv vytvoriť súbor, do ktorého sa bude zapisovať defragmentovaný súbor.

Vytvorenie cesty k novému súboru

Výslednú cestu by si, samozrejme, mal možnosť vybrať užívateľ, ak tak neučiní, názov súboru sa vytvorí automaticky nasledujúcim spôsobom. Vstupným parametrom bude cesta k pôvodnému súboru a výstupným parametrom bude vytvorená cesta tak, že sa za názov súboru pridá text `_defrag`, ktorý bude informovať o tom, že tento súbor je defragmentovanou kópiou pôvodného súboru. Pôvodná cesta k súboru sa bude stále končiť príponou `.tdms`, ktorá bude mať stále rovnakú dĺžku piatich znakov. Tieto znaky sa vymažú a následne opäť doplnia textom `_defrag.tdms`. Ak sa takýto súbor už v priečinku vyskytuje, je vhodné dať užívateľovi možnosť vybrať, či chce tento súbor prepísať.

Taktiež by mala byť na výber aj možnosť, či sa po defragmentácii pôvodný súbor vymaže, aby zbytočne nezaberal priestor na disku.

Táto funkcia by však nemala byť užívateľským prostredím, preto je vhodné tieto možnosti riešiť spôsobom vstupných parametrov.

Ak sa funkcia neukončí z dôvodu existencie defragmentovaného súboru alebo chyby, proces pokračuje otvorením oboch príslušných súborov.

Aby pri práci so súborom nedošlo k poškodeniu dát, je vhodné pôvodný súbor otvoriť len pre čítanie a nový súbor len pre zápis. Pre istotu je možné nastaviť ukazovateľ pôvodného súboru na jeho začiatok, aby sa prečítal celý súbor.

Následne sa môže začať s procesom čítania pôvodného súboru, ktorý však môže byť podstatne optimalizovaný oproti tomu, ako je opísaný v kapitole 4.

5.1 Optimalizácia čítania súboru

Optimalizácia uvedená v tejto kapitole môže byť čiastočne prevedená aj pre klasické čítanie TDMS súboru, uvedené v kapitole 4.

Taktiež ako bolo uvedené v podkapitole 3.1, TDMS súbory s DAQmx Raw dátami sa vyznačujú štruktúrou, pri ktorej sa defragmentácia nevykonáva. Preto ak sa pri čítaní vyskytne segmentem, ktorý obsahuje DAQmx Raw dáta, defragmentácia sa ukončí s chybou. To nám umožňuje odstrániť niektoré časti kódu:

- **čítanie indexu Raw dát pre DAQmx dáta**
- **ukladanie vlastností s mierkami z predošlého segmentu**
- **čitanie DAQmx Raw dát**
- **prepočet DAQmx Raw dát podľa mierok**

Zjednodušiť je možné taktiež štruktúru, ktorá bola v predošlej kapitole nevyhnutná kvôli celkovému zobrazeniu TDMS formátu. Zo štruktúry teda môžeme odstrániť tieto časti:

- **cestu súboru**
- **počet segmentov**
- **pozície jednotlivých častí v súbore**
- **TDMs tag**
- **ofsety na ďalší segment**
- **ofsety na Raw dáta**
- **štruktúra DAQmx Raw dáta indexu**
- **pole kanálov**

Prípadne je možné odstrániť aj ďalšie časti, ktoré je možné kedykoľvek dopočítať, ako sú hodnoty vyjadrujúce dĺžky všetkých ciest a názvov a dĺžka indexu Raw dát.

V prostredí LabVIEW je možné taktiež odstrániť premenné s počtom objektov a počtom vlastností, pretože si polia ukladajú údaje o svojich veľkostiach automaticky.

Tieto premenné musia byť, samozrejme, zo súboru prečítané, ale ďalej pri čítaní sa nepoužívajú, preto ich nie je potrebné ukladať.

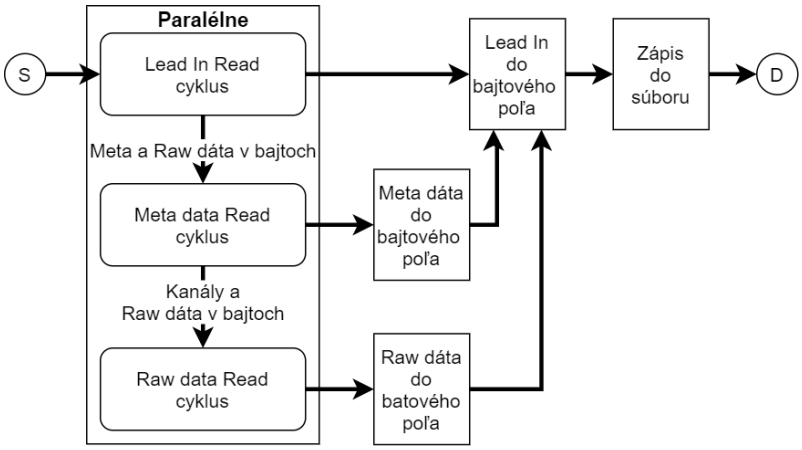
5.1.1 Paralelizovanie procesu

Väčšina dnešných procesorov obsahuje viacero vlákien, ktoré dokážu pracovať naraz, čo môže podstatne zvýšiť rýchlosť čítania súboru.

Po odskúšaní rôznych spôsobov je najvýhodnejšie použiť princíp „pipelining“, ktorý bol v prostredí v LabVIEW dosiahnutý pomocou použitia front. Podstatou tohto princípu je jednotlivé procesy rozdeliť čo najrovnomernejšie medzi vlákna procesora, aby trvali približne rovnakú dobu. Ale keďže pri čítaní TDMS súboru závisí dĺžka operácie na štruktúre súboru, ktorá sa môže meniť, je najvhodnejšie procesy rozdeliť aj logicky:

1. **Lead In**
2. **Meta dáta**
3. **Raw dáta**

Takto rozdelené procesy na obr.5.1 pri bežných TDMS súboroch trvajú približne rovnaký čas a sú zároveň logicky rozdelené podľa štruktúry TDMS súboru.



Obr. 5.1: Paralelizovaný diagram algoritmu pre defragmentáciu TDMS súborov.

Dnešné procesory môžu disponovať aj väčším počtom vlákien, ktoré je možné použiť pri paralelizovaní for cyklov pri čítaní Raw dát. Paralelizovanie takýchto cyklov nemusí vždy urýchliť aplikáciu, naopak, môže ju spomaliť, ak sa v cykle vykonáva iba jednoduchý výpočet a doba preposielania dát medzi vláknami prevýši dobu ušetreného času rozdelením výpočtov. Taktiež môžu využívať viac operačnej pamäte, čo pri veľkých súboroch nie je optimálne.

5.1.2 Defragmentácia Meta dát

Tento proces nemusí zvýšiť rýchlosť čítania súboru, práve naopak, pri svojom vykonávaní ho spomaľuje, no je nevyhnuté pre získanie potrebných informácií o celom

súbore. Tento proces sa nemôže vykonávať predtým, než sa prečítajú dáta jedného segmentu, pretože Meta dáta v segmente neobsahujú nutne všetky kanály, ktoré sú v súbore uložené. Po tom, čo sa v Meta dátach aktualizuje zoznam objektov, sa zisťuje, aké objekty naozaj segment obsahuje. Ak segment neobsahuje nový zoznam objektov, čo podľa nastavenia bitu „kTocNewObjList“, požíva sa celý aktualizovaný zoznam objektov. Z tohto zoznamu sa musí najprv vypočítať,koľkokrát sú zapísanie hodnoty kanálov v Raw dátach. Po vykonaní týchto procesov sú hodnoty kanálov totožné s veľkosťami kanálov v Raw dátach.

Potom nasleduje tento proces defragmentácia, kde sa vlastne vytvára zoznam objektov, ktorý sa postupne aktualizuje tak, že sa k parametrom o počte hodnôt/celkovej veľkosti v bajtoch pripočítavajú hodnoty z aktuálneho segmentu. Po ukončení čítania je výsledkom zoznam všetkých objektov, ktoré sa v súbore vyskytli, pričom obsahujú celkový počet hodnôt, ktoré sa v súbore nachádzajú.

Počas procesu sa aktualizujú aj vlastnosti objektov, a to tak, že ak sa názov vlastnosti už v objekte vyskytol, jej hodnota sa prepíše na novú.

Tento proces predlžuje čítanie súboru, no výsledné hodnoty sa dajú použiť pre prípadnú jednorázovú alokáciu pamäte, čo môže iné procesy zase urýchliť.

5.1.3 Optimalizácia čítania Raw dát

Treba si uvedomiť, že pri defragmentácii nie je nutné čítať jednotlivé hodnoty, ale stačí vybrať z načítaných Raw dát pole bajtov, ktoré je určené na prekopírovanie do defragmentovaného súboru. To však neplatí pri dátovom type string, pri ňou je potrebné rozdeliť pole bajtov na pole ofsetov a na pole znakov. Taktiež pri zápise do súboru je potrebné pole ofsetov prepočítať, aby odpovedali reálnej pozícii stringov.

Pre ostatné dátové typy sa môže čítanie Raw dát zjednodušiť, ak všetky kanály majú rovnaký počet hodnôt a sú rovnakého dátového typu, z čoho vyplýva, že majú aj rovnakú veľkosť v bajtoch. To je možné iba použitím 2D poľa pri vytváraní súboru pomocou funkcie TDMS Write. 2D pole je nutné použiť pri zapisovaní dát s rozložením interleaved. Taktiež ak sa hodnota ToC masky „TocNewObjList“ rovná hodnote logickej „0“, tiež môže nastať prípad, kde kanály budú mať rovnaké veľkosti v bajtoch aj napriek rozdielnym dátovým typom, napríklad, ak súbor bol už defragmentovaný a znovu použitý pre ukladanie dát.

Najlepším spôsobom je spočítať si veľkosti všetkých kanálov počas čítania objektov v Meta dátach pomocou rovnice.

$$veľkosť\ kanálu = počet\ hodnôt\ x\ veľkosť\ dátového\ typu \qquad (5.1)$$

Vo verzii TDMS súboru 2.0 nie je nutné rovnicu 5.1 násobiť hodnotou „dimenzia poľa“ v indexe Raw dát, pretože je stále rovná 1. Po pridaní rovnice ale nie je nutné pred výpočtom zisťovať verziu súboru. Ak je v indexe Raw dát prítomná hodnota „celková veľkosť v bajtoch“, tak veľkosť kanálu je táto hodnota. [2]

Z predošlej kapitoly 4 vieme, že Raw dáta sa môžu vyskytovať viacnásobne. Teda koľkokrát sú dáta kanálov zapísané, zistíme pomocou rovnice 5.2

$$x = \frac{ofset\ na\ ďalší\ segment - ofset\ na\ Raw\ dáta}{veľkosť\ Raw\ dát\ všetkých\ kanálov} \tag{5.2}$$

kde x je výsledná hodnota pre počet zápisov v Raw dátach. [2]

Takto sa dá aj vopred pripraviť pole, v ktorom budú veľkosti jednotlivých kanálov, čo urýchli čítanie Raw dát. Zároveň nie je potrebné pracovať s cyklami, ale stačí použiť nástroje pre prácu s poľami, ako napríklad transponovanie. Celkovo tak pre optimálnu rýchlosť čítania Raw dát vyplýva 5 možností:

1. jeden kanál
2. rovnaké veľkosti kanálov - non-interleaved
3. rovnaké veľkosti kanálov - interleaved
4. rôzne veľkosti kanálov - non-interleaved
5. rôzne veľkosti kanálov - interleaved (môže nastať iba pre string)

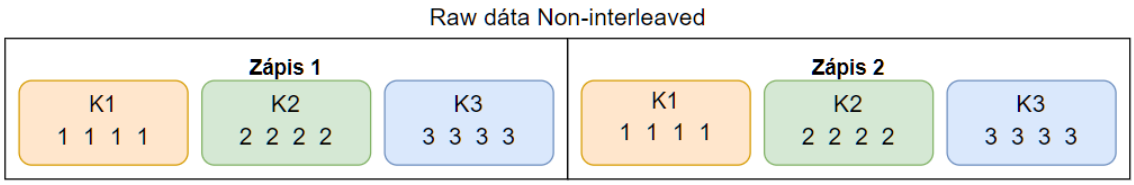
Pre každý typ spracovania je prečítané 1D pole bajtov, práca s ním sa však pri každom type odlišuje kvôli optimalizácii rýchlosti. Pre ilustráciu sú príklady vysvetlené s hodnotami. Pri práci v súbore je potrebné rozdeliť 1D pole na 2D pole hodnôt. Zmení si iba to, že sa bude pracovať s poľami o dimenziu väčšími.

Jeden kanál

V tomto prípade je spracovanie naozaj jednoduché a nie je potrebné nijak pole upravovať, nezáleží na spôsobe jeho uloženia (interleaved/non-interleaved) a ani na tom, koľkokrát je kanál v Raw dátach zapísaný, stačí ho len uložiť k príslušnému kanálu v poli kanálov.

Rovnaké veľkosti kanálov - non-interleaved

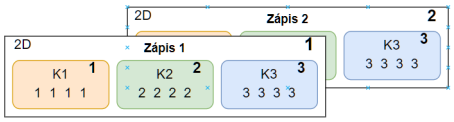
Tento proces musí, samozrejme, fungovať aj pre jeden kanál, ale sú v ňom zbytočné operácie, ktoré sa pri jednom kanály vykonávať nemusia. Pre lepšie vysvetlenie operácie je na obr. 5.2 zobrazené vstupné pole non-interleaved Raw dát, zložené z troch kanálov, ktoré obsahujú 4 hodnoty. Tieto kanály sú v Raw dátach dvakrát. Pod ním je aj na obr. 5.5 zobrazené ako má vyzerať výsledné pole, rozdelené do jednotlivých kanálov.



Obr. 5.2: Raw dáta zapísané v súbore pre 3 kanály s opakovaným zápisom.

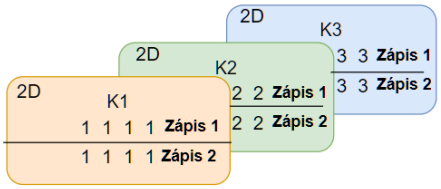
Z obrázku obr. 5.2 sa dá pochopiť, že ak by bol súbore iba jeden zápis, polia by už boli rozdelené do svojich kanálov a nebola by potrebná žiadna operácia. Ak je však v súbore viac zápisov, je potrebné vykonať nasledovné operácie:

1. pretvoriť pole na 3D s dimenziami (zápisy, kanály, hodnoty)



Obr. 5.3: Pretvorené pole na 3D.

2. transponovať výsledné pole podľa horných dimenzií (kanály, zápisy)



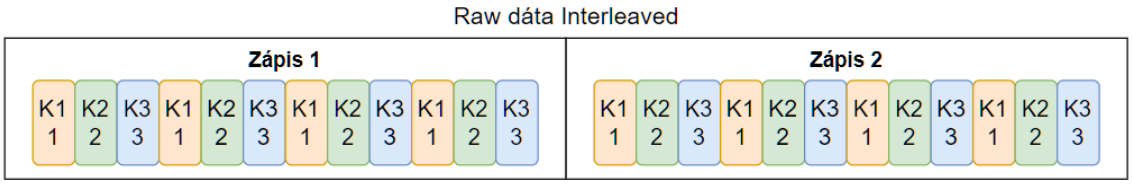
Obr. 5.4: Transponované 3D pole.

3. pretvoriť na 2D pole kanálov (kanály, zápisy x hodnoty)

Obr. 5.5: Výsledné polia rozdelené do svojich kanálov.

Rovnaké veľkosti kanálov - interleaved

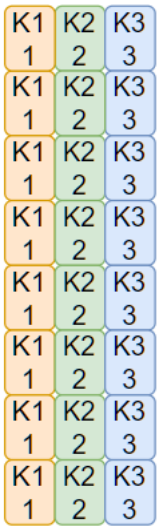
Aj v tomto prípade pre lepšie pochopenie je vstupné pole Raw dát na obr.5.6. Požadovaný výsledok je znova na obr.5.5.



Obr. 5.6: Interleaved Raw dáta zapísané pre 3 kanály s opakovaným zápisom.

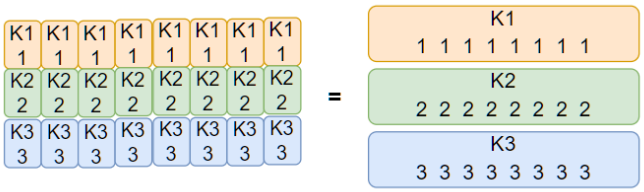
Je možné na prvý pohľad vidieť, že pole nie je úplne závislé na počte zápisov a pri jednom zápise sa úprava nijak nezjednoduší. Preto je v každom prípade nutné vykonať tieto operácie:

1. pretvoriť pole na 2D s dimenziami v poradí (zápisy x hodnoty, kanály)



Obr. 5.7: Interleaved pretvorené 2D pole.

2. transponovať 2D pole



Obr. 5.8: Transponované 2D pole.

Rôzne veľkosti kanálov - non-interleaved

V týchto prípadoch už nie je možné používať len operácie pre polia, ale je nutné použiť aj cykly. Použijú sa vnorené for cykly, kde vonkajší je pre počet opakovaných zápisov a vnútorný je pre počet kanálov. Postupne vo vnútornom cykle prechádzame kanály a podľa ich veľkosti sa oddeľujú z poľa Raw dát. Z oddelených kanálov sa vytvárajú samostatné 1D polia. Vonkajší for cyklus opakuje čítanie kanálov, z čoho vznikne 2D pole, ktoré stačí transponovať.

Rôzne veľkosti kanálov - interleaved

Tento prípad je najzložitejší a najpomalší, ale pretože interleaved dáta vznikajú iba za pomoci 2D polí, rozdielne veľkosti jednotlivých kanálov je možné dosiahnuť iba pre dátový typ string.

Načítanie kanálov funguje podobne ako v predošlom prípade, len je potrebné oddeliť časť s ofsetmi pre string a prečítať samotné stringy, aby sa s nimi dalo pracovať. Po ukončení čítania všetkých kanálov a pred opakovaním vonkajšieho cyklu je nutné kanály transponovať. Pred transponovaním sa musia prerobiť do 2D polí ako v prípade, keď boli veľkosti kanálov rovnaké.

Tento proces je nutné vykonať samostatne pre hodnoty a pre stringy.

Funkcia TDMS Write, sa pri vytváraní takéhoto typu súboru ukončí s varovaním, pretože pôvodné nástroje ho nemusia vedieť otvoriť, ak majú kanály rôzne dĺžky.

5.1.4 Ukladanie defragmentovaného súboru

Keď sa celý súbor prečíta, je možné všetky nazbierané dáta pretvoriť na pole bajtov a zapísať do súboru, keďže ak zapisujeme naraz veľký blok dát, je to rýchlejšie.

Lead In

Verzia súboru a bytové poradie sa môžu skopírovať z predošlého súboru. Podľa tohto bytového poradia sa môže pretvárať zostatok Lead In a Meta dáta. Doplnenie ToC masky je jednoduché. Meta dáta s určitosťou existujú a Raw dáta sa overia pomocou veľkosti poľa. Ak nie je prázdne, nový súbor bude Raw dáta obsahovať. DAQmx tag nemôže nastať, pretože defragmentácia nie je možná pre tento typ dát. Veľkosti ofsetov sa zistia po pretvorení ostatných častí na bajtové polia.

Meta dáta

Pomocou cyklov pre objekty a vlastnosti podľa TDMS formátu sa defragmentované Meta dáta pretransformujú na pole bajtov a zistí sa veľkosť tohto poľa.

Raw dáta

Pomocou cyklu for sa postupne prechádzajú kanály z Meta dát, ktoré sú uložené v správnom poradí. Z poľa kanálov, ktoré sa vytvorili pri čítaní Raw dát, sa postupne vyberajú iba kanály rovnaké s kanálom z Meta dát, čím sa získa dátové pole pre celý jeden kanál. Toto sa opakuje pre každý kanál v Meta dátach a bajtové polia sa potom ukladajú za sebou.

V prípade dátového typu string treba najprv upraviť pole ofsetov, a to tak, že sa posledná hodnota z predošlej časti kanála pripočíta, k všetkým hodnotám ofsetov a následne sa uložia do bajtového poľa v poradí ofsety a pole stringov.

5.1.5 Problém s operačnou pamäťou

V predošlom paralelnom rozložení nastáva problém, ktorý neumožňuje zápis do súboru počas práce s ním, pretože bez kompletne defragmentovaných Meta dát nie je možné vedieť budúcu pozíciu Raw dát v súbore, opísané v podkapitole 5.1.2. Preto je nutné držať všetky defragmentované dáta počas čítania v operačnej pamäti, čo značne obmedzuje veľkosť čítaného súboru na násobky veľkosti operačnej pamäte. Záleží na tom, akým spôsobom bolo do súboru zapisované. Ak pre každý segment je zapísaná iba jedna hodnota, bude mať defragmentovaný súbor niekoľkonásobne menšiu veľkosť.

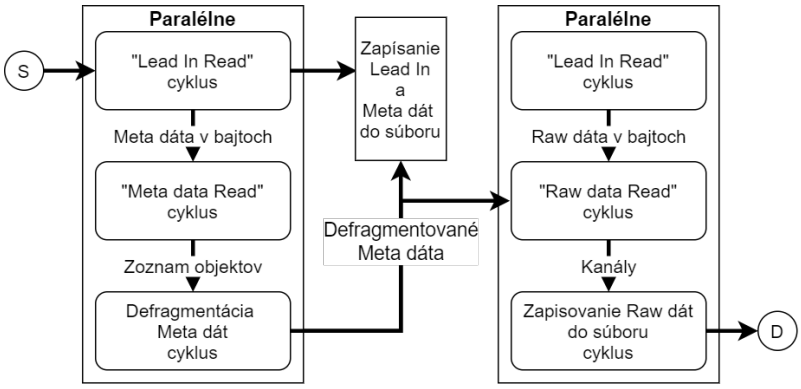
Tento spôsob je ale rýchlejší, a preto môže mať nejaké využitie. V knižnici je pomenovaný ako „TDMS Defragment In Memory“.

5.1.6 Úprava procesu defragmentácie

Aby bola defragmentácia použiteľná aj pre veľké súbory, je nutné upraviť jej vonkajšiu štruktúru, pričom vnútorné funkcie ostanú takmer rovnaké. Pre zachovanie rýchlosti čítania je znovu dobre prácu rozdeliť na viac vlákien procesora.

Hlavným rozdielom oproti predošlému rozloženie je to, že Meta dáta sa musia načítať prvé, a až potom je možné čítať Raw dáta, ktoré sa dajú do súboru postupne zapisovať. Tento zápis by mal byť v samostatnom vlákne, aby nespomaľoval celkový proces.

Uvedeným požiadavkám vyhovuje diagram algoritmus na obr.5.9.



Obr. 5.9: Upravený diagram algoritmu pre paralelizovanú defragmentáciu TDMS súborov s postupným zapisovaním do súboru.

V tomto tvare chýbajú postupné informácie o veľkosti Raw dát v jednotlivých segmentoch, preto ich treba nejako získať. Bolo by možno nevýhodné využívať nové vlákno pre opätovné čítanie Meta dát, to záleží aj na počte vlákien procesora. Ak by sa Meta dáta znovu čítali, vnútorné funkcie by nepotrebovali takmer žiadnu úpravu, až na funkciu Lead In, ktorá by raz čítala zo súboru Meta dáta, a potom Raw dáta. Bolo by možné použiť aj rovnakú slučku s malými úpravami.

Ak by sa Meta dáta čítali iba raz, v každom objekte by okrem počtu hodnôt a celkovej veľkosti v bajtoch pre dátový typ string museli byť aj polia, ktoré by ukladali nasledovné parametre pre každý segment:

- počet hodnôt
- celková veľkosť v bajtoch (string)
- počet zapísaní

V parametroch be tiež mohla byť prepočítaná hodnota o jednotnej veľkosti segmentov, ktorá by určovala postup čítania Raw dát, uvedeného v podkapitole 5.1.3.

Postupné ukladanie dát

Cyklus pre ukladanie dát nemá za úlohu len zapisovanie do súboru, ale aj indexovanie prečítaných kanálov v poradí, v akom boli načítané v súbore, podobne ako je vysvetlené v podkapitole 5.1.4.

Ukladanie sa nemusí vykonávať v každom cykle, ale hodnotu o veľkosti dát si môže program ukladať a zapísať do súboru až po presiahnutí určitej veľkosti.

Zápis jednotlivých kanálov musí byť na vopred určené ofsety pomocou defragmentovaných Meta dát. Následne sa tieto ofsety zväčšujú pre každý kanál samostatne, pretože kanály sú na iných pozíciách, aby mali počas zapisovania dostatok priestoru. Pred každým zápisom do súboru sa nastaví ukazateľ na hodnotu ofsetu a k tomuto ofsetu sa pripočíta hodnota veľkosti zapísaných dát.

Takto sa dosiahne, aby sa operačná pamäť nezaplnila pri nadmerne veľkých súboroch.

Pre zapisovanie Lead In a Meta dát sa nič nemení.

6 Nástroj pre generovanie chýb v TDMS súboroch

Aby sme dokázali vyskúšať funkcionalitu nástrojov pre opravu TDMS súborov, potrebujeme najprv TMDS súbory s rôznymi chybami. Oprava TDMS súboru závisí nielen od toho, aká chyba nastala, ale aj kde nastala. Preto je potrebné rozdeliť chyby nielen podľa typu, ale aj podľa miesta v súbore, kde nastali.

Tento nástroj bude schopný generovať 3 typy chýb, ktoré si bude môcť užívateľ zvoliť:

- vynechanie určitého počtu bajtov
- zmena určitého počtu bajtov
- simulácia predčasného ukončenia zápisu (napr. výpadkom napájania)

Ako už bolo spomenuté, záleží aj na mieste výskytu chyby, tú bude možné zvoliť podľa čísla segmentu a podľa toho, v akej časti sa nachádza (Lead In, Meta data, Raw data), opísané v podkapitole 2.1 a ofsetom pre danú časť. Najprv sa pôvodný súbor prečíta, aby sme mohli určiť počet segmentov a maximálnu hodnotu ofsetu pre každú časť.

6.1 Realizácia generovania chýb

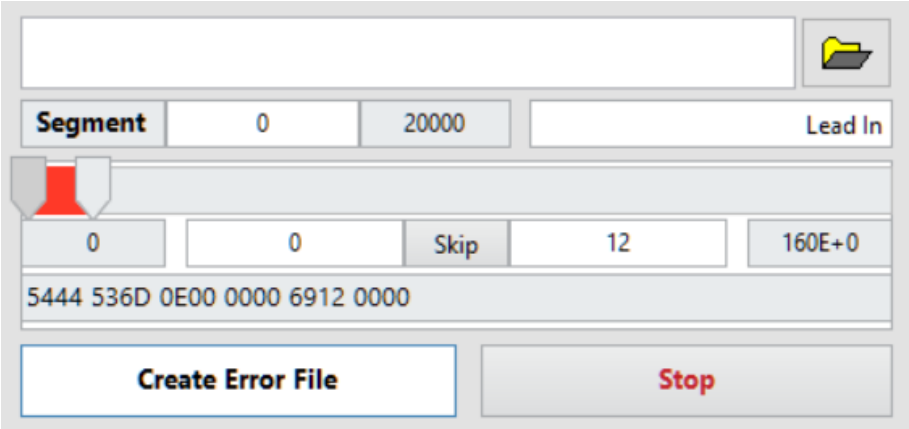
Používateľ si najprv vyberie vlastnosti chyby, program môže spustiť a vygeneruje sa pre danú chybu špecifický názov podľa počtu segmentov, dátového typu a vlastností chýb, aby bolo z názvu jasné, s akým súborom pracuje. Pred výberom typu chyby sa vypočíta pozícia v súbore, kde sa má chyba nachádzať, a to pomocou ofsetov v Lead In z konkrétneho segmentu a vybraného ofsetu používateľom.

Vynechanie určitého počtu bajtov Vytváranie tejto chyby v programe bude nasledovné. Pôvodný súbor sa kopíruje až pokiaľ sa nedosiahne vybraná pozícia, následne sa pozícia v súbore posunie o daný počet bajtov a pokračuje v kopírovaní súboru.

Zmena určitého počtu bajtov Tento typ chyby sa vytvára podobným spôsobom ako pri predošlej chybe tak, že pôvodný súbor sa kopíruje po vybranú pozíciu s tým rozdielom, že namiesto pôvodných bajtov sa vloží rovnaký počet náhodných bajtov a tie sa zapíšu do súboru. Ako v predošlej chybe sa pokračuje v kopírovaní s ofsetom o rovnakej veľkosti bajtov.

Predčasné ukončenie zápisu Pri tomto type sa jednoducho skopíruje súbor iba po vybranú pozíciu.

Užívateľské prostredie nástroja na obr. 6.1, obsahuje vrátane častí uvedených v tejto kapitole aj časti, ktoré pomáhajú užívateľovi lepšie vybrať miesto poruchy ako je napríklad zobrazenie poľa bajtov, na ktorých bude vygenerovaná porucha vybraného typu.



Obr. 6.1: Užívateľské prostredie nástroja pre generovanie chýb v TDMS súboroch v prostredí LabVIEW.

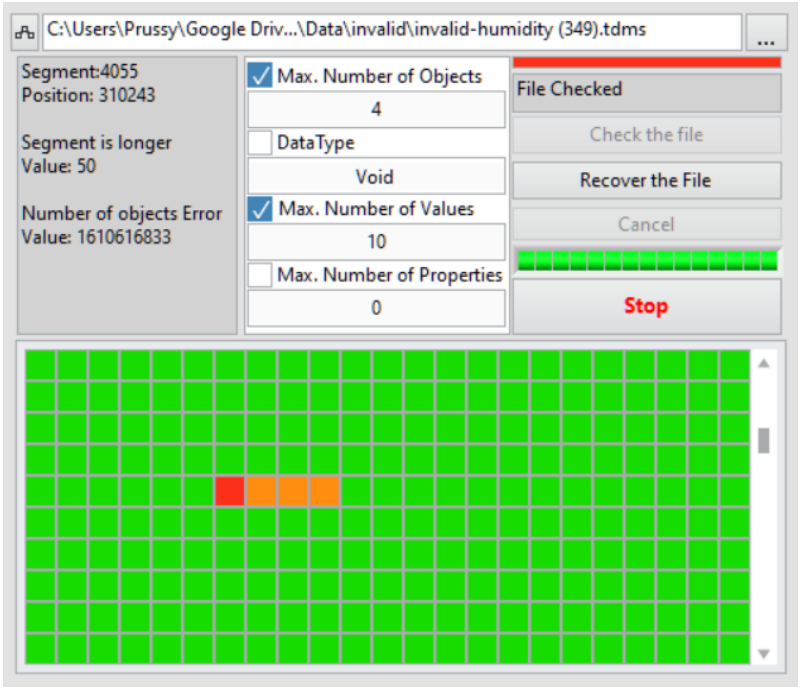
7 Nástroj pre získavanie dát z chybných súborov

Keďže formát súboru v sebe nemá zakomponované žiadne redundantné informácie, pomocou ktorých by bolo možné zistiť správnosť formátu súboru, je možné že pre plnú funkcionalitu bude potrebné doplnenie určitých informácií užívateľom pre odhalenie a opravu niektorých chýb.

Program bude rozdelený na funkcie, ktoré budú pracovať s TDMS súborom, a na časť užívateľského rozhrania, pomocou ktorého sa budú môcť sprostredkovať užívateľské vstupy a výstupy. Rozdelenie zabezpečí možnosť použitia funkcií aj v iných programoch.

Nástroj má v sebe zakomponované 3 funkčné procesy, ktoré bude môcť užívateľ spúšťať postupne pomocou užívateľského rozhrania:

1. **skúška súboru** - po zadaní cesty k súboru sa vyskúša jeho spustiteľnosť a či je možné súbor opraviť.
2. **kontrola chýb súboru** - ak súbor je poškodený a dá sa opraviť, užívateľ môže spustiť kontrolu chýb, ktorá zistí, kde sa v súbore chyba nachádza. Následne nástroj graficky zobrazí štruktúru súboru.
3. **obnovenie súboru** - podľa výberu užívateľa sa nástroj pokúsi vytvoriť obnovený súbor.



Obr. 7.1: Užívateľské prostredie pre funkciu „TDMS_Recovery“, na kontrolu a získavanie dát z poškodených súborov, vytvorenú v prostredí LabVIEW.

Užívateľské prostredie uvedené na obr. 7.1 v sebe zahrňuje všetky 3 spomenuté procesy, a zároveň poskytuje užívateľovi informácie o chybných segmentov v podobe farebných štvorčekov znázorňujúcich jednotlivé segmenty. Po kliknutí na štvorček sa užívateľovy zobrazia informácie o segmente vrátane typu chýb, ktoré sa pri kontrole odhalili.

7.1 Skúška správnosti súboru

Funkciou tejto časti je zistiť, či daný súbor je naozaj súbor TDMS, následne sa kontroluje, či nie je natoľko poškodený, že sa nedá opraviť, a to získaním jeho veľkosti a porovnaním s jej najmenšou možnou veľkosťou TDMS súboru. Zistí sa to tak, že si vytvorí čo najjednoduchší TDMS súbor pomocou oficiálnych nástrojov, a to jednoduchým vytvorením TDMS súboru pomocou funkcie TDMS Open. Tento súbor má veľkosť 69 bajtov. Ak bude mať kontrolovaný súbor menšiu alebo rovnakú veľkosť dá sa usúdiť, že zo súboru sa nedokážu získať žiadne zmysluplné dáta.

Ak súbor splňuje tieto požiadavky, potom ho program vyskúša otvoriť pomocou pôvodných nástrojov a tak zisti či je súbor poškodený. Ak pri otváraní nastane chyba, program vie, že tento súbor je poškodený a užívateľské prostredie môže užívateľovi poskytnúť možnosť kontroly chýb súboru.

7.2 Kontrola chýb súboru

Tento proces má za úlohu nájsť všetky možné príčiny, prečo TDMS súbor nedokáže byť otvorený normálnymi nástrojmi. Kontrolu integrity dát nie je možné urobiť, pretože TDMS formát v sebe nemá zabudovanú žiadnu redundanciu (kontrolný súčet, parita). Preto sa každá hodnota musí kontrolovať, či je v určitom rozmedzí hodnôt, určenom približne výpočtami alebo zadané užívateľom.

Proces vykonáva kontrolu postupne po segmentoch a zisťuje, v ktorom segmente nastala chyba. Taktiež si zaznamenáva pozíciu segmentu v súbore, aby sa vedel pri oprave k segmentu vrátiť. Ak sa v segmente nájde chyba, proces sa preruší, uloží si chybu a hodnotu premennej, pri ktorej nastala. Takto má potom užívateľ prehľad, akou príčinou chyba nastala, prípadne riešenia, ako ju opraviť.

V súbore sa taktiež môžu objaviť chyby, ktoré si nevyžadujú prerušenie kontroly, ale naopak, pre získanie ďalších informácií o chybe je vhodné v kontrole pokračovať. Tieto chyby nie sú kritické pre ďalšie čítanie, lebo indikujú chybu v inej časti súboru.

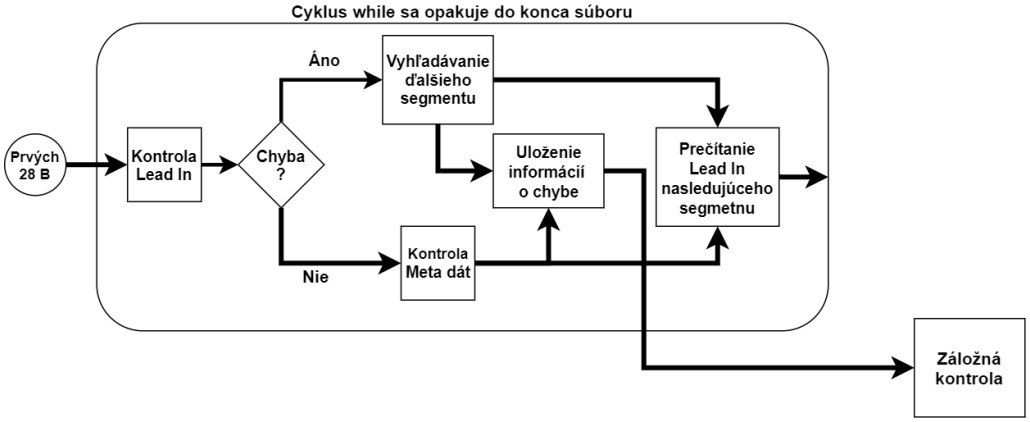
Po skontrolovaní celého súboru sa užívateľovi zobrazí pole segmentov, ktoré sa v súbore našli. Farebne sú odlíšené chybné segmenty od správnych. Po kliknutí na

určitý segment sa vypíše, v akom segmente nastala chyba, pozícia segmentu v súbore, názov chýb a hodnota premennej.

Proces je rozdelený na 2 hlavné časti (Kontrola Lead In, Kontrola Meta dát) z dôvodu väčšej prehľadnosti a prípadne možnosti paralelizovať tento proces pre optimalizáciu rýchlosti spracovania.

Raw dáta sa nekontrolujú z dôvodu obtiažnosti ich kontroly, pretože táto kontrola by mohla byť založená iba na užívateľsky zadanom rozmedzí. Taktiež chyba v Raw datách nemusí znemožniť spustenie súboru.

Na obr. 7.2 je zobrazený diagram algoritmu pre kontrolu chýb v TDMS súboroch, pričom jednotlivé bloky sú opísané v nasledujúcej časti podkapitoly.



Obr. 7.2: Diagram algoritmu pre kontrolu chýb v TDMS súboroch.

7.2.1 Kontrola Lead In

Z predošlej kapitoly 2.1 je známy formát Lead In, ktorý obsahuje 28 bajtov. Preto si na začiatku každého segmentu zo súboru prečítame 28 bajtov, ktoré sa budú kontrolovať.

Každú premenná sa kontroluje, až pokiaľ sa nezistí, že hodnota je chybná. Ak sa nezistí žiadna chyba, program sa pokúsi o skočenie v súbore na nasledujúci segment, skontroluje, či neprekročil veľkosť súboru alebo či sa nenachádza na jeho konci. Ak sa veľkosť súboru prekročí, vyhodnotí sa to ako chyba. Taktiež sa skontroluje, či sa na danej pozícií nachádza „TDSm tag“ pre kontrolu počiatku segmentu.

Ak chyba nastane pri kontrole hodnôt, ďalej sa Lead In nekontroluje a taktiež nie je známa pozícia nasledujúceho segmentu. Výnimka nastáva pri chybe Raw dáta ofsetu. Keďže v hierarchii pred ním sa nachádza ofset na ďalší segment môže sa prejsť na čítanie ďalšieho segmentu, ale z dôvodu, že tieto ofsety sú úzko spojené, je veľká pravdepodobnosť, že ak je chybný jeden ofset, tak je chybný aj druhý.

Taktiež TDMS súbor musí mať niektoré informácie v Lead In jednotné v každom segmente. To sú informácie o poradí bytov a verzie TDMS súboru. Tieto informácie sa za normálnych okolností pri práci s pôvodnými nástrojmi nedajú v jednom súbore meniť.

Preto je taktiež potrebné porovnávať prvý segment súboru s ostatnými, a ak sa nájdu rozdiely v týchto informáciách, nepovažuje sa to za fatálnu chybu súboru, keďže sa predpokladá, že ostatné dáta v segmente sú nepoškodené, len inak uložené.

Preto pri akejkoľvek z týchto chýb sa proces preruší a začne sa vyhľadávať nasledujúci segment.

Vyhľadávanie nasledujúceho segmentu

Vyhľadávania nasledujúceho segmentu pri chybe v Lead In sa začína s ofsetom jeden bajt za počiatkom predošlého segmentu pre precízne skontrolovanie súboru.

Pred začiatkom hľadania je potrebné vybrať vhodný počet bajtov, ktorý sa bude čítať, aby sa nemusel načítavať do pamäte celý zbytok súboru naraz. Tento počet sa dokáže určiť približne z priemernej veľkosti predošlého segmentu. Ten sa vypočíta z pozície v súbore a jej vydelením počtom segmentov, ktoré sa doteraz prečítali. To sa dokáže zistiť z toho, v akej iterácii cyklu kontroly sa program nachádza. Ak sa kontroluje prvý segment, tak veľkosť sa môže určiť z celkovej veľkosti súboru vydelenou nejakou vhodnou konštantou (napr. 10).

Toto pole bajtov sa následne prehľadáva v cykle, kde porovnáva 4 bajty s „TDSm tagom“, aby sa našiel začiatok ďalšieho segmentu. Keďže sa prehľadávajú 4 prvky poľa, cyklus sa musí ukončiť o 3 iterácie skôr, ako je veľkosť poľa aby sa v poli stále nachádzal dostatok prvkov.

Je možné, že sa v načítanom poli bajtov „TDSm tag“ nenájde, preto je vhodné tento proces vyhľadávania vložiť do cyklu **while**. Keďže vnútorný cyklus sa ukončí predčasne, musia sa zvyšné 3 prvky dostať na začiatok ďalšieho načítaného poľa aby sa nevynechala žiadna časť súboru. Následne tento cyklus musí byť ukončený dvoma podmienkami:

1. nájdenie TDSm tagu nasledujúceho segmentu
2. koniec súboru

Preto sa musí kontrolovať, či sa ukazovateľ nenachádza na konci súboru, aby sa pri načítaní ďalšieho poľa bajtov nepresiahla veľkosť súboru. Ak bude odhadnutý počet bajtov väčší ako počet bajtov do konca súboru, musí program načítať iba tento daný počet bajtov, aby nenastala pri čítaní zo súboru chyba.

Ak sa zhoda s „TDSm tagom“ nájde, program ukončí oba cykly, z ich počtu iterácií a veľkosti odhadnutého poľa bajtov vypočíta pozíciu nasledujúceho segmentu v súbore.

Ak chyba nebola v nesprávnych hodnotách pri kontrole Lead In, a predsa počiatok nasledujúceho segmentu sa nenachádzal na určenom mieste, predpokladá sa, že chyba sa nachádza v zbytku segmentu a preto je ho potrebné skontrolovať, aby sa zistilo, kde sa chyba nachádza.

Keďže hodnoty ofsetov dokážu byť kontrolované len približne, nemusia byť správne, preto je pre kontrolu Meta dát vhodnejšie načítať pole od konca predošlého Lead In až po začiatok nasledujúceho, s ktorým už bude pracovať Funkcia Kontrola Meta dát. Taktiež je potrebné načítať Lead In pre kontrolu nasledujúceho semgentu.

7.2.2 Kontrola Meta dát

K tejto kontrole sa program dostane iba v prípade, ak segment nejaké Meta dáta obsahuje a zároveň v časti Lead In nenastala žiadna kritická chyba, keďže pre kontrolu Meta dát sú potrebné tieto informácie, ktoré sa prečítali vo funkcií Kontrola Lead In.

Kontrola Meta dát len podľa údajov z Lead In nemusí byť až taká presná a nedokáže odhaliť všetky chyby v súbore, preto pri odhalení chýb môžu pomôcť manuálne zadanie maximálnych parametrov od užívateľa, ktoré si užívateľ môže nastaviť, a tak presnejšie určiť rozmedzie, v ktorom sa určité údaje majú nachádzať.

Užívateľské parametre pre kontrolu

- počet objektov
- dátový typ
- počet nameraných hodnôt
- počet vlastností

Užívateľ si môže zvoliť, ktoré z týchto parametrov sa majú alebo nemajú použiť pre kontrolu, aby nedošlo k označeniu správnych segmentov pri nenastavených parametroch.

Kontrola jednotlivých hodnôt

Pre každú hodnotu sa dá vypočítať približná maximálna hodnota, ktorú môže nadobudnúť v konkrétnom segmente. Kontrola niektorých hodnôt vo formáte TDMS je samozrejmá a nepotrebuje ďalšie vysvetlenie, preto nie sú uvedené všetky hodnoty.

ToC maska musí nadobúdať hodnotu menšiu ako je `0x EE` a zároveň ak má maska aktívnu jednu z hodnôt „kTocInterleavedData“ alebo „kTocDAQmxRawData“, musí mať aktívnu aj hodnotu „kTocRawData“.

Ofset na ďalší segment sa musí rovnať 0, ak segment neobsahuje Raw alebo Meta dáta. Ak súbor obsahuje tieto dáta, tak x[1] $< veľkosť\ súboru - 28$ (dĺžka Lead In).

Ofset na Raw dáta sa musí rovnať ofsetu na ďalší segment, ak segment neobsahuje Raw dáta. Ak Raw dáta obsahuje, musí byť menší.

Počet objektov sa nesmie rovnať 0 ak segment obsahuje Meta dáta a zároveň musí platiť $x < (veľkos\ poľa\ dát - 4)/13$(minimálna veľkosť objektu)[2] $+ 1$.

Dlžka cesty objektu nesmie byť rovná 0 a zároveň $x < zvyšné\ bajty - (zvyšné\ objekty \times 13)$.

Cesta objektu musí mať prvý znak „/“ a všetky znaky zároveň v intervale <31,126>.

Dĺžka indexu Raw dát $x = \{20, 28\}$.

Počet vlastností $x < \frac{zvyšné\ bajty - (zvyšné\ objekty \times 13)}{10(minimálna\ veľkosť\ vlastnosti)}$[3].

Dĺžka názvu vlastnoti sa nesmie rovnať 0 a zároveň $x < (zvyšné\ bajty - (zvyšné\ objekty \times 13) - zvyšné\ vlastnosti \times 10)$.

Zvyšne bajty znamenajú chybu pri čítaní Meta dát.

Nedostatok bajtov sa zistí, ak pri konverzii na dátový typ nastane chyba z dôvodu malej veľkosti poľa bajtov.

Ukladanie dát o chybách v segmente

Aby nielen užívateľ, ale aj samotný program mal informácie o súbore a chybách, je potrebné si ukladať po kontrole jednotlivých segmentov určité parametre.

Pri každom segmente je potrebné vedieť, či v ňom sa v ňom nachádza chyba, ale aj aký typ chyby. Na to je vhodné si vytvoriť dátový typ **enum**, v ktorom budú zahrnuté všetky druhy chýb, ktoré môžu nastať, vrátane bezchybného stavu, aby aj užívateľ mohol vidieť, aká chyba nastala, prípadne upraviť užívateľské parametre. Taktiež je vhodné ukladať ku každej chybe jej prislúchajúcu chybnú hodnotu.

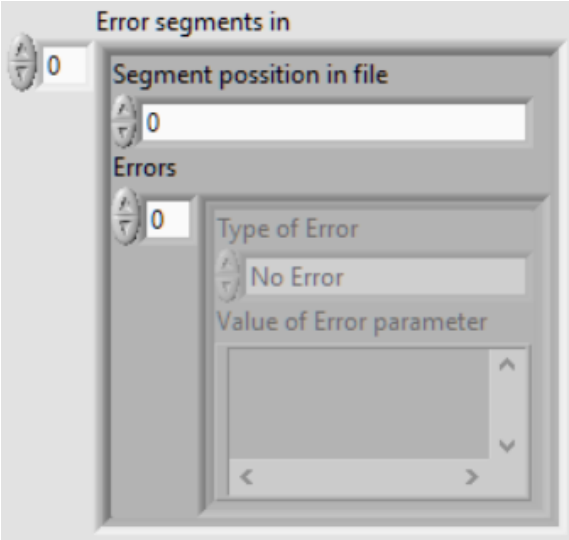
Ako bolo vysvetlené v predošlých podkapitolách 7.2.1 a 7.2.2, môžu nastať prípady, keď pri kontrole Lead In nenastala fatálna chyba a pokračuje sa v kontrole Meta dát. To znamená, že jeden segment môže obsahovať viacej chýb, čiže pre každý segment musí existovať pole chýb a ich príslušných chybných hodnôt.

Pre jednoduchšiu orientáciu pri ďalšom procese opravy súboru je taktiež vhodné ukladať ďalšie informácie, ako je pozícia v segmente, kde chyba nastala a taktiež veľkosť chybného segmentu, na čo je postačujúca jedna hodnota pozície počiatku alebo konca segmentu. (V tejto práci sa bude pokračovať s parametrom o pozícii počiatku segmentov)

[1] x pri kontrole hodnôt je stále opisovaná hodnota

[2] Hodnota 13 je stále hodnota pre minimálnu veľkosť objektu

[3] Hodnota 10 je stále hodnota pre minimálnu veľkosť vlastnosti



Obr. 7.3: Názorná štruktúra pre ukladanie chýb súbora v prostredí LabVIEW.

7.2.3 Záložná kontrola chybného súboru

Pri kontrole predošlým spôsobom sa nemusia odhaliť všetky chyby súboru, napríklad ak je poškodená len časť cesty k objektu, ale dĺžka cesty a jednotlivé znaky cesty sú stále v rozmedzí validných znakov. To môže viesť k tomu, že sa v danom súbore, ktorý neide otvoriť, nenájde žiadna chyba. Preto v predošlej metóde kontroly sa musí zaznamenať, či tento stav nenastal a následne vykonať záložnú kontrolu súboru.

Záložná kontrola súboru spočíva v tom, že program si vytvorí nový dočasný TDMS súbor, do ktorého postupne zapisuje jednotlivé segmenty a následne kontroluje, či súbor je možné otvoriť pôvodnými funkciami v LabVIEW. Takto zistí, ktorý segment zo súboru je chybný a bude ho možné obnoviť.

Dôvodom, prečo tento spôsob kontroly sa nerobí primárne je, že táto metóda je omnoho pomalšia ako predošlá metóda opísaná v podkapitolách 7.2.1 a 7.2.2, pretože musí zapisovať do súboru na disku. Táto metóda taktiež nedokáže odhaliť bližšie informácie, kde chyba nastala, a hlavne vďaka predošlým uloženým pozíciám počiatku segmentov 7.2.2 program dokáže oveľa rýchlejšie prekopírovať daný segment do dočasného kontrolného súboru.

Táto metóda nie je veľmi vhodná pre veľké súbory nielen preto, že môže proces zabrať veľa času, ale aj preto, že veľmi veľké súbory sa nemusia zmestiť na disk, preto je lepšie nechať zvoliť užívateľa, či chce vykonať túto kontrolu a upozorniť ho na to, že môže trvať podstatne dlhší čas.

7.3 Získavanie dát zo súboru

Celá kontrola súboru sa vykonala len k získaniu potrebných informácií o chybe, pričom najpodstatnejšia informácia je index segmentu, v ktorom sa chyba nachádza. Ak má súbor veľa segmentov, je jednoduchšie a rýchlejšie odstrániť chybné, a tak s malými stratami na dátach obnoviť celý súbor. Použitie tohoto procesu pri súbore s jedným segmentom by znamenalo, že by sa z neho nezískali žiadne dáta. Pre typy TDMS súborov s malým počtom segmentov je vhodné vytvoriť poloautomatizovaný proces obnovy súboru, čím sa zaoberá celá podkapitola 7.4.

7.3.1 Výber obnovenia užívateľom

O výbere z týchto dvoch možností by mal rozhodnúť užívateľ podľa poskytnutých informácií o počte segmentov a typu chýb, ktoré v súbore nastali. Ak užívateľ zvolí možnosť pre opravu segmentov, mal by tiež poskytnúť pomocné informácie o vytváraní súboru, ktoré sú jednotné pre celý súbor:

- podobnosť segmentov (či bol súbor vytváraný pomocou cyklov)
- v poškodenej časti sa môže nachádzať viacej segmentov

Prípadne môže doplniť aj informácie o dátovom poradí, verzií, prítomnosť DAQ Raw dát alebo usporiadaní dát (inverleaved/non-interleaved). Informácie o maximálny parametroch a dátovom type sa taktiež môžu zísť pri procese opravy, ak boli pred tým zadané užívateľom.

7.3.2 Kopírovanie bezchybných segmentov

Táto metóda je najjednoduchšia a najrýchlejšia pre získanie dát zo súboru, no dokáže získať iba obmedzené množstvo dát.

Jej princípom je postupné kopírovanie bezchybných segmentov do nového súboru, pokým sa nevyskytne chybný segment, ktorý sa jednoducho preskočí. Pri tomto kopírovaní môžeme použiť predom uložené pozície jednotlivých segmentov 7.2.2.

Pri tejto metóde môže ale nastať problém, kde segmenty, ktoré sú bezchybné sa nachádzajú bezprostredne za chybným segmentom a neobsahujú nový zoznam objektov, čiže niektoré informácie sa čerpajú z predošlých segmentov. Bližšie popísané v podkapitole 4.2.3.

Tento problém je najlepšie odhaliť už pri kontrole súboru a treba označiť tieto segmenty. Aby sa nevytvárala nijaká nová premenná, je možné si vytvoriť nový typ chyby (**Matching Raw data index**), ktorá sa bude samostatne sledovať pri kopírovaní segmentov.

V niektorých prípadoch sa tento problém nemusí prejaviť nespustiteľnosťou súboru alebo sa môže prejaviť len čiastočne, chybnými hodnotami v segmente. Preto

je vhodné dať na výber užívateľovi, či chce, aby sa program pokúsil o kopírovanie týchto segmentov a zároveň ho upozorniť, že hodnoty vo výslednom súbore môžu byť chybné.

Následne sa pri ich kopírovaní bude kontrolovať spustiteľnosť nového TDMS súboru. Ak by nebol spustiteľný, je potrebné posledný segment odstrániť a pokračovať v kopírovaní.

7.4 Oprava chybných segmentov

Ak si užívateľ vyberie túto možnosť, oprava bude naďalej prebiehať kopírovaním nepoškodených segmentov do nového súboru, pokiaľ sa nevyskytne chybný segment, až potom sa nástroj pokúsi o opravu tohto segmentu.

Predtým než sa začne oprava segmentu, je potrebné získať čo najviac informácií o súbore. Najvalidnejšie informácie sa nachádzajú v predošlých segmentoch, ktoré sa už nachádzajú v novom súbore. Tento súbor je možné celý prečítať pomocou vytvorenej funkcie **TDMS_Read** v kapitole 4. Nesmie sa však zabudnúť na to, že jeho ukazovateľ ukazuje na koniec súboru. Potom čo sa ukazovateľ prestaví na počiatok a celý prečíta, sa znovu dostane na pôvodnú pozíciu.

Ak chyba nastala v prvom segmente, nie je možné získať z predošlých žiadne informácie , preto je optimálne sa pokúsiť získať informácie z nasledujúceho segmentu. Jeho počiatočná pozícia by mala byť známa, pretože je potrebná pre určenie veľkosti aktuálneho segmentu. Tento segment sa však nachádza v pôvodnom súbore, preto je nutné po jeho prečítaní nastaviť ukazovateľ súboru na pôvodnú hodnotu. Čítanie sa prevedie, iba ak segment neobsahuje chybu v Lead In alebo v Meta datách a to pomocou samostatných funkcií **Lead_In_Read** a **Meta_data_Read**, opísaných v podkapitolách 4.1 a 4.2.

V neposlednom rade je možné získať časť informácií aj z aktuálneho chybného segmentu, ktoré sa budú pri postupnej oprave doplňovať. Jeho prečítanie nám umožní uložená hodnota o pozícii chyby, o ktorej sa písalo v podkapitole 7.2.2. Pomocou nej je možné segment rozdeliť na poškodenú a nepoškodenú časť.

Nepoškodenú časť je možné taktiež prečítať pomocou funkcií Lead In read a Meta data read s malými úpravami. Rozdelením segmentu sa taktiež dosiahlo, že prí čítaní sa nebudú nachádzať chybné hodnoty, ktoré by mohli spôsobiť závažné chyby pri zle ošetrenom algoritme. Samozrejme, že prí čítaní nastanú chyby, ktoré sa však môžu odignorovať.

V súbore sa často môže vyskytovať jeden chybný segment, preto je nutné pred opravou vyhodnotiť, aké dáta sa budú používať. Najvhodnejšie je, ak sú dostupné všetky tri spomenuté možnosti.

7.4.1 Vyhľadávanie validných dát

Aj keď TDMS formát nemá v sebe zabudované redundantné informácie, vyskytujú sa ňom určité údaje pri ktorých sa dá s veľkou istotou určiť, že tvoria štruktúru súboru:

- **TDMS verzia**
- **cesty objektov**
- **index Raw dát**
- **názov vlastnosti**

Medzi údaje patrí aj TDSm tag, o vyhľadávanie ktorého sa už program raz pokúšal, takže sa v segmente nenachádza.

Akou metódou sa začne, sa určí podľa typu chyby, presnejšie pri akej hodnote zo štruktúry TDMS súboru obr.4.1 sa ukončilo čítanie súboru.

TDMS verzia Táto metóda sa zvolí, ak chyba nastala pred čítaním TDMS verzie, čo znamená pri čítaní hodnôt TDSm tagu alebo ToC masky. Ak sa poškodenie súboru vyskytlo len vo veľmi krátkom úseku, je možné, že boli poškodené len tieto hodnoty a za nimi pokračuje nepoškodená časť súboru.

Zo skúmania dodaných poškodených súborov sa zistilo, že sú často chybné viaceré segmenty, takže ak sa aj nájde táto hodnota, tak je veľmi pravdepodobné, že sa v poškodenom úseku nachádza viac segmentov a aj to, že nejaké nepoškodené dáta sa môžu nachádzať pred pozíciou nájdených bajtov TDMS verzie.

Ak sú dostupné dáta o verzii TDMS súboru a jeho bajtovom poradie, je jednoduché zvoliť hodnotu, ktorá sa bude v súbore vyhľadávať. Ak tieto dáta nie sú dostupné, je nutné postupne vyskúšať vyhľadávanie všetkých variant:

- 0x00001269 = 0d4713 (2.0)
- 0x69120000 = 0d1762787328
- 0x00001268 = 0d4712 (1.0)
- 0x68120000 = 0d1746010112

Tieto hodnoty sú však v istej miere rovnaké s hodnotami index DAQmx Raw dát. Ich rozlíšenie je uvedené v časti pre metódu indexu Raw dát 7.4.1.

Cesty objektov Je to najefektívnejšia metóda zo všetkých vymenovaných, čo sa týka vyhľadávania a aj rekonštrukcie dát, pretože po nej nasledujú všetky informácie o zápise Raw dát.

Metódu je možné použiť kedykoľvek, no optimálne je ju zvoliť ako počiatočnú v rozmedzí chýb na parametroch **<TDMS Verzia, Počet objektov>** a tiež pri parametroch: **názov,dátový typ a hodnota vlastnosti** kvôli vysokej pravdepodobnosti výskytu ďalšieho objektu.

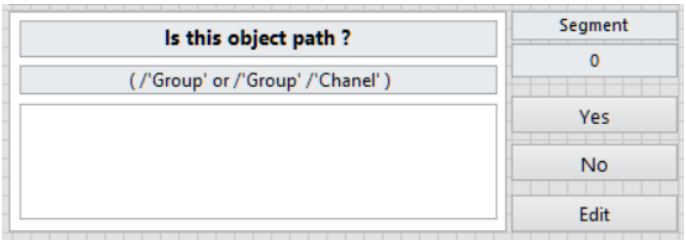
Cesta objektov má stále rovnaké počiatočné znaky "/’" pre skupinu a kanál a "/" pre objekt súboru. Ten sa vyskytuje výhradne v prvom segmente. Pred začatím vyhľadávania ciest objektov je vhodné najprv zistiť, či sa tieto znaky v poli vôbec nachádzajú. Pre prvý segment sa kontrola vykoná so znakom "/", ak sa neopravuje prvý segment je lepšie použiť znaky "/’".

Ak sa znaky nachádzajú v prvom segmente, je vhodné vykonať ešte jedno vyhľadávanie, a to znakov "**/FFFFFFFF**", čo je cesta objektu súboru s indexom Raw dát, ktorý má pri tomto objekte stále hodnotu 0xFFFFFFFF, pričom nie je potrebné ani vedieť bajtové poradie.

V prípade ďalších segmentov alebo ak znaky neboli nájdené sa pokračuje vyberaním objektov na vyhľadávanie z nadobudnutých dát a to takým spôsobom, že sa porovnajú cesty objektov z ostatných segmentov s cestami aktuálneho segmentu, ak boli nejaké prečítané. Ak nie sú dostupné dáta z ostatných segmentov, z dostupného segmentu je možné oddeliť z cesty pre kanál iba cestu skupiny, ktorá je rovnaká pre ostatné kanály.

Ak ani v tomto prípade sa žiadne zhody nenašli, je možné, že v segmente boli vytvorené nové kanály, pretože podľa prvého vyhľadávanie znakov "/" a "/’" je možné usúdiť, že sa v segmente nachádzajú minimálne časti ciest.

Pri ďalšom procese je nutné, aby užívateľ posúdil, či nájdené znaky sú alebo nie sú aspoň sčasti cestami objektov, a v prípade znalosti ich doplnil. Užívateľovi sa po nájdení znaku "/’" zobrazí časť užívateľského prostredia, zobrazená na obrázku 7.4 aj so zvyškom cesty, ktorá môže byť aj upravená pomocou predošlých 4 bajtov, ktoré by mali vo formáte TDMS obsahovať dĺžku danej cesty. Hodnotu je možné zistiť iba so znalosťou bajtového poradia a následnej kontroly. Hodnota musí byť menšia ako dĺžka zvyšku poľa bajtov a zároveň väčšia ako 4, pretože objekt okrem objektu súboru môže mať takýto minimálny tvar "/’<1 znak>’", čo sú minimálne 4 znaky. Ak týmto požiadavka hodnota, nevyhovuje pole znakov sa neoreže.



Obr. 7.4: Dialógové okno pre overenie cesty k objektu v prostredí LabVIEW

.

Počas skúmania dodaných chybných súborov pomocou tejto metódy sa zistilo, že cesty k objektom často obsahujú korektné časti názvov skupiny alebo kanálu,

ale dáta v ich okolí boli stále natoľko poškodené, že sa nedalo pokračovať v oprave segmentu.

Index Raw dát Touto metódou sa dajú automaticky kontrolovať iba určité hodnoty index Raw dát, keďže niektoré ich číselné hodnoty sa mohli vytvoriť v súbore následkom poruchy.

Najlepšie rozlíšiteľná hodnota indexu Raw dát je hodnota pre DAQmx Raw dáta, ktorá je však v niektorých prípadoch rovnaká s hodnotou verzie TDMS, uvedené v tab.7.1

Hodnota	Verziaa TDMS	Index DAQmx Raw dát
4712	1.0	Analogové DAQmx Raw dáta
4713	2.0	Digitálne DAQmx Raw dáta (uvedená v lit.[2])
4714	-	Digitálne DAQmx Raw dáta (reálne v súboroch)

Tab. 7.1: Porovnanie hodnôt, ktoré môže nadobudnúť verzia TDMS a index DAQmx Raw dát.

Rozlíšiť je ich možné tak, že po indexe DAQmx Raw dát bezvýhradne nasleduje hodnota dátového typu DAQmx Raw dát `0x FF FF FF FF`, na rozdiel od verzie TDMS súboru, za ktorou nasleduje hodnota ofsetu na ďalší segment, ktorej správnosť nie je možné s určitosťou zistiť. Z toho je možné usúdiť, že ak sa za nájdenou hodnotou z tab.7.1, nachádza hodnota `0x FF FF FF FF` a súbor obsahuje DAQmx Raw dáta, ide o index Raw dát. Ak nie, hodnota vyjadruje verziu TDMS súboru, ktorá musí byť rovnaká s verziami ostatných segmentov.

Z poškodených súborov sa zistilo, že sa málo vyskytuje aj samotná hodnota `0x FF FF FF FF`, ktorá hovorí o tom že objekt neobsahuje Raw dáta. Vyhľadávanie tejto hodnoty je lepšie až potom, čo sa našla hodnota indexu DAQmx Raw dát a súbor tento typ dát neobsahuje. Aby sa dalo automaticky povedať, že ide naozaj o hodnotu indexu Raw dát, načítajú sa ďalšie 4 bajty, ktoré by mali obsahovať počet vlastností. Správnosť tejto hodnoty sa dá určiť s pomocou približného výpočtu maximálnej hodnoty uvedenej v rovnici 7.1 alebo s pomocou potvrdenia od užívateľa.

$$maximálny\ počet\ vlastností = \frac{veľkosť\ zvyšného\ poľa}{minimálna\ veľkosť\ vlastnosti\ (10\,B)} \tag{7.1}$$

Ďalšou hodnotou index Raw dát je jeho dĺžka, ktorá sa v segmente vyskytuje, ak ide o nový objekt. Táto hodnota môže nadobúdať len 2 stavy:

- dátový typ string (28)
- ostatné dátové typy (20)

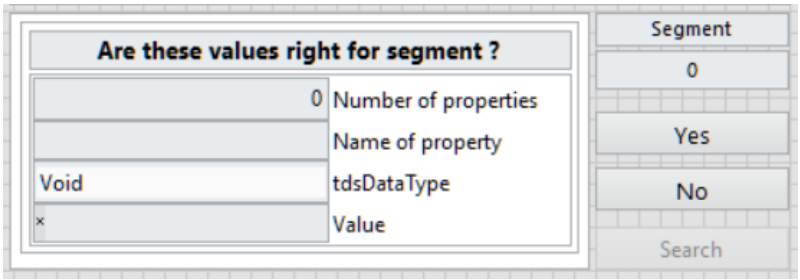
Aj tu sa dá hodnota skontrolovať prečítaním ďalších 4 bajtov, ktoré by mali obsahovať dátový typ, čiže môže nadobúdať hodnoty uvedené v enume tdsDataType uvedenom v podkapitole 2.1.5. Prípadne ešte ďalšou hodnotou s rovnakou veľkosť, určujúcou dimenziu poľa, ktorá je vo TDMS verzii 2.0 stále rovná 1.

Hodnota indexu Raw dát pre opakujúci sa objekt `0x 00 00 00 00` nie je vhodná pre automatické vyhľadávanie, pretože sa vyskytuje v súbore často aj bez toho aby bol nejak poškodený. Toto vyhľadávanie je možné iba za pomoci užívateľa, ktorý musí určiť, či hodnoty, ktoré sa nachádzajú za hodnotou `0x 00 00 00 00`, sú možnými hodnotami pre tento segment.

Ak je táto hodnota naozaj indexom Raw dát, musí za ňou nasledovať, ako aj v prípade žiadnych Raw dát, hodnota o počte vlastností. Počet vlastností je vo väčšine prípadov nulový, preto môže byť zložité pre užívateľa určiť správnosť tejto hodnoty, keďže sa vyskytujú v súbore často. Je nutné teda prečítať a zobraziť užívateľovi aj nasledujúce hodnoty, ktoré môžu byť rôzne. Aké parametre by mali vo formáte nasledovať, dokáže sa určiť práve podľa počtu vlastností:

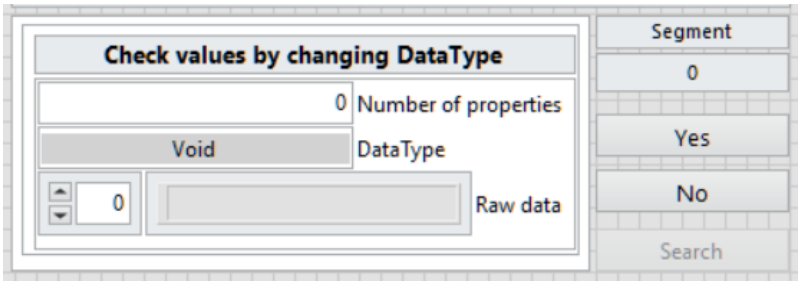
- počet vlastností > 0
- počet vlastností $= 0$

Je samozrejmé, že ak sa nachádzajú v objekte vlastnosti, mali by podľa formátu aj nasledovať, preto sa užívateľovi zobrazí zoznam parametrov vlastnosti, ako je ukázané na obr. 7.5.



Obr. 7.5: Užívateľské prostredie pre kontrolu hodnôt v prostredí LabVIEW.

V prípade ak je počet vlastností nulový, môžu nastať zase 2 prípady. Vo formáte je uvedený ďalší objekt alebo nasledujú Raw dáta. Ak sa kontrola ciest objektov vykonáva pred touto metódou, je možné úplne vylúčiť, že sa za nulovým počtom vlastností nachádza ďalší objekt, ktorý nie je poškodený. Potom je vhodnejšie načítať zvyšné pole bajtov a predať užívateľovi, ktorý si vyberie očakávaný dátový typ, a skontroluje či dané hodnoty môžu byť hodnotami Raw dát. Táto časť užívateľského prostredia je vyobrazená na obr. 7.6.

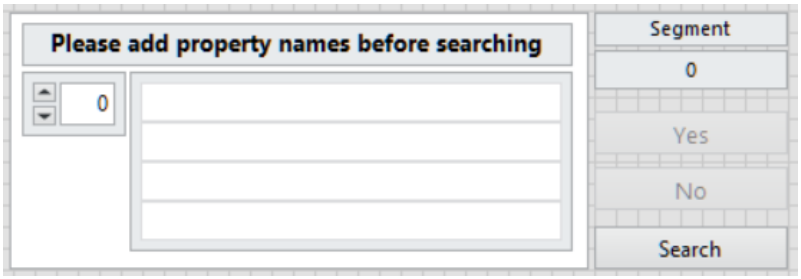


Obr. 7.6: Užívateľské prostredie pre kontrolu správnosti Raw dát s výberom dátového typu v prostredí LabVIEW.

Názov vlastnosti Vlastnosti sa v súboroch s veľkým počtom segmentov často nevyskytujú, preto má táto metóda len malú úspešnosť. Je ale veľmi jednoduchá a dokáže určiť polohu vlastností s vysokou istotou.

Jedna vlastnosť sa však vyskytuje v každom súbore, ale iba v prvom segmente súboru. Táto vlastnosť má stále rovnaký tvar v súbore. Názov vlastnosti je **„name“** a jej hodnota **„<názov súboru>“**. Ak sa tento názov v prvom segmente nenájde, je možné skúsiť vyhľadať aj hodnotu vlastnosti, ktorú sa dá získať z cesty súboru oddelením prípony `.tdms` a zvyšku cesty pomocou posledného znaku „\“.

V ostatných segmentoch, pri automatizovanom riešení sa najprv prehľadajú dostupné segmenty a ich názvy vlastností sa uložia do poľa. Následne sa skontroluje výskyt každého názvu vlastnosti v poškodenom segmente. Ak aj tento proces zlyhá, je nutné, aby užívateľ dodal zoznam názvov vlastností v užívateľskom prostredí, ktorého časť pre vyhľadávanie vlastností je zobrazená na obr. 7.7 pre prostredie LabVIEW.



Obr. 7.7: Časť užívateľského prostredia pre doplnenie názvov vlastností v prostredí LabVIEW.

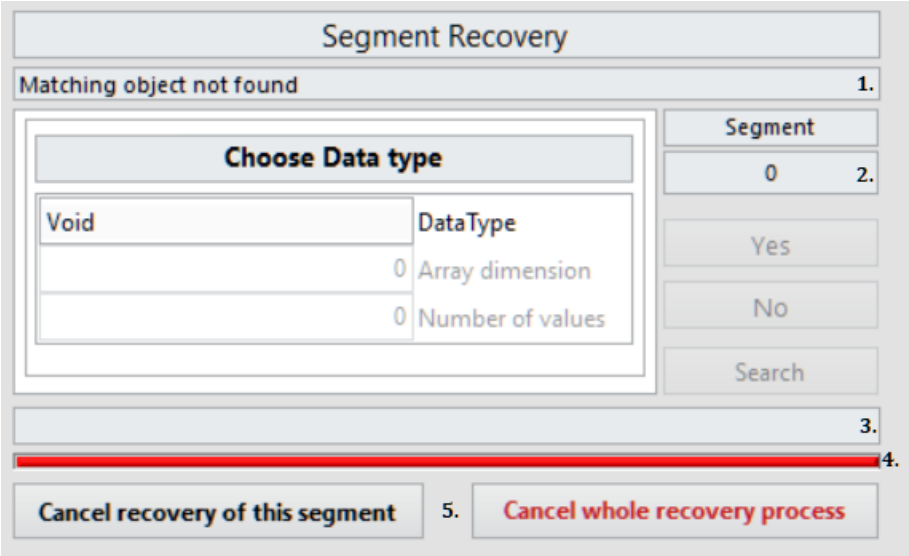
7.4.2 Vyhľadávanie užívateľom

Užívateľské prostredie nie je vhodné volať samostatne pri každej jednej metóde opravy, ale je lepšie vyskúšať segment, opraviť najprv za pomoci automatizovaných

spôsobov, aby nebol užívateľ príliš obťažovaný. Až potom, čo automatizovaný spôsob zlyhá, je vhodné zavolať užívateľské prostredie, kde sa užívateľ pokúsi o opravu ručne pomocou vyššie uvedených metód.

Celkové užívateľské prostredie, ktoré je zobrazené na obr.7.8, obsahuje všetky predošle spomenuté časti. Prostredie je zároveň doplnené aj o rôzne pomocné informácie, ktoré umožnia užívateľovi lepšie analyzovať chybu v súbore:

1. správy o procese vyhľadávania
2. segment, ktorý sa prehľadáva
3. zvyšné pole bajtov
4. ukazovateľ pozície v chybnom segmente s farebnou signalizáciou úspešnosti vyhľadávania
5. možnosti pre ukončenie procesu opravy



Obr. 7.8: Celé užívateľské prostredie pre manuálne prehľadávanie chybného segmentu v prostredí LabVIEW.

7.4.3 Analýza chybných súborov

Predtým než sa opíše rekonštrukcia segmentov, je potrebná určitá analýza poškodených súborov, ktorá pomôže objasniť, ako zvyčajne vyzerajú poškodené segmenty. Analyzované súbory v prílohách boli vytvárané zapisovaním v cykle, kde sa zapisovanie opakovalo každých 5 sekúnd. Nepoškodené súbory obsahujú 20000 segmentov, kde pri jednom zápise sa do súboru zapísali 4 segmenty. Čas zapisovania do súboru si následne vypočítame pomocou rovnice 7.2.

$$t = \frac{Počet\ segmentov}{4} \times 5 = \frac{20000}{4} \times 5 = 25000[s] \sim 7[h] \tag{7.2}$$

Pomocou tohto nástroja sa pri kontrole viacerých poškodených súborov zistilo, že približný počet nepoškodených segmentov je 18000, čo znamená cca 2000 poškodených segmentov. Pomocou rovnice 7.2 je možné znovu vypočítať, že proces chybného zapisovanie trval cca 40 minút.

Ďalej pri kontrole pomocou nástroja na vyhľadávanie validných dát sa vo väčšine prípadov nepodarí nájsť validné dáta. Pri niektorých súboroch boli validné dáta na úplnom konci tohto chybného zväzku segmentov.

Z toho vyplýva, že väčšina porúch pri zapisovaní do súborov trvá dlhšiu dobu, teda nedostatočne krátku na to, aby sa poškodila drobná časť segmentu. A taktiež ak sa pri oprave súboru vyskytnú validné dáta pre viac než len zopár hodnôt formátu, je veľmi pravdepodobné, že zvyšok segmentu je nepoškodený.

Táto úvaha zjednoduší čítanie Raw dát z chybného segmentu a zároveň len minimálne obmedzí možnosť opravy pre TDMS súbory, kde sa vyskytla málo pravdepodobná porucha.

7.4.4 Rekonštrukcia segmentu

Rekonštrukcia segmentu sa môže začať, iba ak sa našli validné dáta a ich pozícia v chybnom segmente. Ak sa žiadne validné dáta nenašli, segment sa považuje za neobnoviteľný.

Nájdené validné dáta nám potom určujú počiatok čítania súboru, ktoré pokračuje od pozície validných dát, až pokiaľ sa nenarazí na ďalšiu chybu alebo na predpokladaný koniec Meta dát, čiže nulový počet vlastností alebo posledná vlastnosť. Nezávisle na tom, či je informácia o počte objektov v aktuálnom segmente známa, sa musí vykonať kontrola ďalších parametrov, pretože v prehľadávanom chybnom segmente sa môže vyskytovať viacej poškodených segmentov, ktorých hodnoty „TDSm tag“ boli zmenené a program neodhalil ich výskyt. To znamená, že pozícia validných dát môže ukazovať na segment s odlišným zoznamom objektov.

Preto sa najprv urobí jednoduchá kontrola pre výskyt nasledujúceho objektu, kde podľa formátu TDMS sú najprv uložené 4 bajty, a následne cesta objektu začínajúca znakom „/“. Čiže sa skontroluje, či bajt v poli je týmto znakom. Potom sa môže pokračovať v čítaní ďalšieho objektu podľa TDMS formátu, kde sa znovu monitoruje, či nenastala žiadna chyba.

Ak sa na tejto pozícii nenachádza znak „/“, pokračuje sa kontrolou Raw dát. Táto kontrola môže byť vykonaná iba pomocou užívateľa, a to dvoma spôsobmi. Ak je dostupný dátový typ pomocou načítaných predošlých segmentov, tohto segmentu alebo zadané užívateľom a interval v akom sa môžu hodnoty Raw dát vyskytovať, môže program rozhodnúť o správnosti dát automaticky. Ak nemá prístup k týmto informáciám, vyzve užívateľa pre manuálnu kontrolu s výberom dátového typu.

Nesprávne hodnoty dáta v tomto prípade znamenajú, že segment je neopraviteľný. Je ale možné, že sa ďalšie segmenty nachádzajú v zbytku chybného bajtového poľa, preto pre zvýšenie šancí na opravu segmentu sa môže celý proces opakovať od vyhľadávania validných dát, opísaných v podkapitole 7.4.1, až pokiaľ sa nenájdu žiadne validné dáta. Tento proces sa môže opakovať, aj keď nastane chyba v rekonštrukcií segmentu s tým, že je zase vhodné určiť akou metódou vyhľadávania dát sa začne.

Čítanie Raw dát

Podľa predošlej analýzy uvedenej v podkapitole 7.4.3, je možné usúdiť, že ak veľkosť poľa Raw dát nemá rozmery, ktoré by odpovedali veľkosti kanálov, dátového typu, alebo počtu hodnôt, zhluk chybných segmentov stále pokračuje. Preto je vysoko pravdepodobné, že v nasledujúcich Raw dátach bude väčšina hodnôt poškodených. Ak je nutné zo súboru dostať čo najviac, je možné sa pokúsiť o čítanie Raw dát s kontrolou, či sa hodnoty vyskytujú v zadanom intervale. Proces sa ukončí po prekročení intervalu.

Ak sa Raw dáta zdajú validné, čiže ich veľkosť je napr. deliteľná veľkosťou dátového typu, počtom hodnôt alebo počtom kanálov, poskytnú sa údaje užívateľovi, aby prípadne doplnil chýbajúce informácie, ktoré sa zo segmentu nedali vyčítať.

Ukladanie opravených segmentov

Opravené objekty v segmente je lepšie ukladať ako nové objekty, aj keď by boli uvedené a rovnaké ako v predošlých segmentov. To nám umožní uložiť tieto segmenty do samostatného súboru, aby aj napriek správnosti jedného opraveného segmentu sa nepoškodil celý súbor, keďže ako je často uvádzané v tejto práci, segmenty na seba naväzujú a nie sú to samostatné celky.

Uloženie segmentu do celého opraveného súboru je možné urobiť, ale následne je potrebné kontrolovať kopírovanie všetkých ostatných segmentov. Ak pri kopírovaní nepoškodeného segmentu sa vytvorí poškodený súbor, je potrebné odstrániť všetky segmenty od opraveného a následne nepoškodené segmenty naspäť nakopírovať. Výsledkom budú 3 súbory: pôvodný, prekopírovaný a súbor obsahujúci opravené segmenty.

Záver

Prvým cieľom tejto bakalárskej práce je podrobné popísanie formátu, v ktorom sa ukladajú TDMS súbory. Keďže tento formát je ukladaný binárne, je práca s ním zložitejšia, ale zato má svoje nesporné výhody, o ktorých hovorí úvodná teoretická časť práce. To sa dosiahlo aj pomocou nástroja pre grafické zobrazenie opísanej štruktúry TDMS formátu. Spolu s implementovaným užívateľským prostredím je tento nástroj možné použiť aj v iných prácach, ktoré by sa zaoberali problematikou spojenou s TDMS súbormi.

Pred praktickou časťou boli opísané obvyklé nástroje, ktoré sa používajú pre prácu s TDMS súbormi a ďalej slúžia ako referencia pri vytváraní knižnice, ktorá je hlavným cieľom tejto práce.

Knižnica okrem spomenutého nástroja pre grafické zobrazenie formátu obsahuje aj funkciu pre defragmentáciu nadmerne veľkých TDMS súborov, pri práci s ktorými môže pôvodná funkcia v LabVIEW zlyhať. Funkcia je implementovaná v knižnici v dvoch verziách. Pri návrhu prvej verzie sa zistilo, že dokáže využívať počas procesu iba operačnú pamäť, čo obmedzuje veľkosť súboru, a tak nevyhovuje požiadavkám. Je ale mierne rýchlejšia a jednoduchšia oproti druhej verzií defragmentácie súboru, ktorá umožňuje postupné zapisovanie na disk, čím splňuje požiadavky na jej funkčnosť pre nadmerne veľké súbory.

Do skupiny špecifických súborov, ktorými sa práca zaoberá, patria aj rôzne poškodené TDMS súbory. Pre generovanie takýchto súboroch sa v knižnici nachádza nástroj s implementovaným užívateľským prostredím pre upresnenie parametrov poruchy.

Po kompletnej analýze TDMS formátu bola navrhnutá a implementovaná podrobná kontrola súborov s týmto formátom, ktorá poskytuje dostatočné informácie pre pokus o opravu súboru, a to dvoma spôsobmi. Prvý spôsob, ktorý vyberie zo súboru iba nepoškodené časti, funguje vo väčšine prípadov a môže byť zároveň plne automatizovaný. Druhý spôsob, ktorý sa pokúša o opravu poškodených častí, je navrhnutý a čiastočne implementovaný aj v knižnici. Nástroj dokáže vyhľadať prípadné miesta, ktoré sú vhodné na opravu, ale nedokáže zo získaného množstva informácií zrekonštruovať dáta poškodenej časti, kvôli komplexnosti TDMS formátu. Po analýze dodaných poškodených súborov sa zistilo, že tento spôsob by dokázal získať naviac oproti prvému spôsobu len minimum dát, preto je implementovaný v knižnici len ako ukážka a pomôcka pre ďalšiu analýzu poškodenia súboru.

Knižnica je plne funkčná, no pre prípadné verejné použitie, by potrebovala vylepšiť spracovanie chýb, ktoré môžu nastať pri používaní užívateľského prostredia.

Literatúra

[1] *What is The Difference Between ASCII and BINARY?* [online]. 23 October, 2015, [cit. 7. 11. 2019]. Dostupné z: <https://www.interserver.net/tips/kb/what-is-the-difference-between-ascii-and-binary/>.

[2] National Instruments: *TDMS File Format Internal Structure* [online]. 23.7.2014. [cit. 7. 11. 2019]. Dostupné z URL: <http://www.ni.com/product-documentation/5696/en/>.

[3] National Instruments: *The NI TDMS File Format* [online]. 19.9.2019.[cit. 7. 11. 2019]. Dostupné z URL: <http://www.ni.com/product-documentation/3727/en/>.

[4] *NI DIAdem* [online]. 2012, [cit. 21. 11. 2019]. Dostupné z URL: <https://www.digitalengineering247.com/pics/0712/05957_DIAdem_Flyer_FG.pdf>.

[5] Jim Hokanson. MATLAB Central File Exchange. *TDMS Reader* [online]. 2011, posledná aktualizacia 20. 12. 2018 [cit. 4. 1. 2020]. Dostupné z URL: <https://mathworks.com/matlabcentral/fileexchange/30023-tdms-reader>

[6] *TDMS python* [online]. 2012, posledná aktualizácia 19. 11. 2019 [cit. 4. 1. 2020]. Dostupné z URL: <https://pypi.org/project/npTDMS/>

[7] *DAQmx Configure Logging (TDMS)* [online]. 2019, Január 2019 [cit. 2020-05-25]. Dostupné z: <shttp://zone.ni.com/reference/en-XX/help/370469AP-01/lvdaqmx/mxlogging/>

Zoznam príloh

A Obsah elektronickej prílohy

V

- / ..... kořenový adresář elektronickej prílohy
  - TDMS Viewer
    - TDMS Viewer.aliases
    - TDMS Viewer.exe
    - TDMS Viewer.ini
  - TDMS Defragment
    - TDMS Defragment.aliases
    - TDMS Defragment.exe
    - TDMS Defragment.ini
  - TDMS Defragment In Memory
    - TDMS Defragment In Memory.aliases
    - TDMS Defragment In Memory.exe
    - TDMS Defragment In Memory.ini
  - TDMS Generate Error
    - TDMS Generate Error.aliases
    - TDMS Generate Error.exe
    - TDMS Generate Error.ini
  - TDMS Recovery
    - TDMS Recovery.aliases
    - TDMS Recovery.exe
    - TDMS Recovery.ini
  - Testovacie súbory
    - TDMS Defragment
      - dbl&i32-little-v2.0-decim.tdms
      - humidity (2295).tdms
      - humidity (2305).tdms
    - TDMS Viewer
      - dbl-big-v2.0-interleave.tdms
      - dbl-little-v1.0-decim.tdms
      - dbl-little-v2.0-decim.tdms
      - string-little-v2.0-decim.tdms
      - wfm-big-v2.0-decim.tdms
    - Poškodené
      - invalid-humidity (349).tdms
      - invalid-humidity (730).tdms
      - invalid-humidity (870).tdms